VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra kybernetiky a biomedicínského inženýrství

# Univerzální SW struktura pro oblast automatizace testování senzorů

# Universal SW Framework for Sensors Automated Test

2019 Bc. Rostislav Pokorný

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra kybernetiky a biomedicínského inženýrství

# Zadání diplomové práce

Student: **Bc. Rostislav Pokorný**

Studijní program: N2649 Elektrotechnika

Studijní obor: 2612T041 Řídicí a informační systémy

Téma: Univerzální SW struktura pro oblast automatizace testování senzorů
Universal SW Framework for Sensors Automated Test

Jazyk vypracování: čeština

Zásady pro vypracování:

Náplní diplomové práce je návrh a realizace modulární platformy pro sjednocení a zefektivnění testování senzorů v automobilním průmyslu. Výsledný nástroj bude implementován ve vývojovém prostředí NI LabVIEW. Součástí řešení práce bude kromě teoretického rozboru i odzkoušení návrhu na generickém senzoru.

1. Úvod do problematiky vytváření aplikací v LabVIEW, možné přístupy k jejich tvorbě.
2. Seznámení se s typickými úlohami v oblasti automatizace testování automobilových senzorů a požadavky na ně kladené. (komunikace s přístroji, komunikace se senzorem, sběr a ukládání dat)
3. Návrh modulární architektury SW pro oblast automatického testování automobilových senzorů.
4. Implementace SW dle návrhu ve vývojovém prostředí LabVIEW.
5. Verifikace a testování SW architektury na vybraném senzoru.
6. Zhodnocení dosažených výsledků.

Seznam doporučené odborné literatury:

[1] VLACH, Jaroslav, Josef HAVLÍČEK a Martin VLACH. *Začínáme s LabVIEW.* 1. vyd. Ilustrace Viktorie Vlachová. Praha: BEN - technická literatura, 2008, 247 s. ISBN 978-80-7300-245-9.
[2] BRESS, Thomas J. *Effective labview programming.* 1st ed. Allendale: NTS Press, 2013, 701 s. ISBN 19-348-9108-8.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **doc. Ing. Petr Bilík, Ph.D.**

Datum zadání: 01.09.2018

Datum odevzdání: 30.04.2019


| | |
|---|---|
| doc. Ing. Jiří Koziorek, Ph.D. | prof. Ing. Pavel Brandštetter, CSc. |
| *vedoucí katedry* | *děkan fakulty* |

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2019 .....

## Prohlášení zástupce spolupracující právnické nebo fyzické osoby

Continental Powertrain Czech Republic s.r.o.
Na rovince 879
72000 Ostrava-Hrabová
Česká republika

Souhlasím se zveřejněním této diplomové práce „Univerzální SW struktura pro oblast automatizace testování senzorů“ dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

Continental
Continental Powertrain Czech Republic s.r.o.
Na Rovince 879, 720 00, Ostrava-Hrabová
IČ: 07574622 DIČ: CZ07574622

V Ostravě dne 30.04.2019

.....
Continental Powertrain Czech Republic s.r.o.

Rád bych na tomto místě poděkoval mému vedoucímu diplomové práce doc. Ing. Petru Bilíkovi Ph.D. za pomoc při vypracování diplomové práce. Rovněž bych chtěl poděkovat zaměstnancům společnosti Continental Powertrain Czech Republic s.r.o., jmenovitě Ing. Janu Coufalovi, Ing. Zdeňku Gerykovi a Ing. Vladimíru Petrovskému Ph.D., kteří mě vedli a poskytovali odborné rady při řešení této práce.

**Abstrakt**

Diplomová práce se zabývá vývojem univerzální SW architektury pro oblast testování a automatizaci měření senzorů. Tato práce vznikla ve Continental Powertrain Czech Republic s.r.o., která vyvíjí a vyrábí automobilové sensory. Práce popisuje teoretický rozbor možných řešení testeru z pohledu návrhových SW vzorů a jaké přinášejí výhody a nevýhody.

Popisem jsou postupné kroky řešení a možné přístupy, které byly zvažovány při vývoji funkčního modulárního a univerzálního SW testeru. V obecném popisu je graficky zobrazen způsob komunikace mezi jednotlivými moduly, použití synchronizačních nástrojů a časování měření ve zvolené architektuře. V praktické části je popis převeden a implementován do funkčního kódu ve vývojovém prostředí NI LabVIEW.

V závěru diplomové práce je vyvinutá architektura SW testeru ověřena na generickém senzoru s vhodnými měřícími přístroji. Měření bylo postaveno tak, aby bylo možné ověřit synchronizaci a následné zpracování reálných zaznamenaných dat, při různém počtu naměřených hodnot a různých vzorkovacích periodách zvolených měřících zařízení.

**Klíčová slova**: LabVIEW, Automobilový průmysl, Senzory, Testování, Návrhové vzory

**Abstract**

The diploma thesis deals with the development of universal SW architecture for the testing area and automation of sensor measurement. This work came into being at Continental Powertrain Czech Republic s.r.o., a company which develops and manufactures automotive sensors. The thesis describes the academic analysis of possible solutions of a tester from the perspective of SW templates and what advantages and disadvantages they bring.

The description is a step-by-step solution and possible approaches that have been considered in developing a functional modular and versatile SW tester. The general description graphically shows the way of communication between individual modules, the synchronization tools usage and timing of measurement in the selected architecture. In the practical part, the description is converted and implemented into functional code in the NI LabVIEW development environment.

At the end of the thesis, the developed SW tester architecture is checked on a generic sensor with suitable measuring devices. The measurement was designed so that it is possible to verify the synchronization and sequential processing of the real recorded datas, with different number of measured values and different sampling periods of selected measuring devices.

**Key Words**: LabVIEW, Automotive, Sensors, Testing, Design Patterns

# Obsah

# Seznam použitých zkratek a symbolů

| | | |
|---|---|---|
| CAN | – | Controller Area Network |
| CNC | – | Computer Numerical Control, Počítačové číslicové řízení |
| DUT | – | Device Under Test, testované zařízení |
| ERS | – | Electronic Reequipment Specification, Požadavky na elektronickou specifikaci |
| GUI | – | Graphical User Interface, Grafické uživatelské rozhraní |
| HW | – | Hardware |
| NC | – | Numerical Control, Číslicové řízení |
| RSD | – | Requirments Specification Document, Dokument požadavků na specifikaci |
| SENT | – | Single Edge Nibble Transmission |
| SW | – | Software |

# Seznam obrázků

# Seznam tabulek

# 1 Úvod

Naplní diplomové práce je vyvinout a realizovat modulární platformu pro sjednocení a zefektivnění testování senzorů v automobilovém průmyslu. Práce byla vypracována ve spolupráci se společnosti Continental Powertrain Czech Republic s.r.o. Jako architektura pro výslednou implementaci bylo zvoleno vývojové prostředí od společnosti National Instruments LabVIEW (Laboratory Virtual Instrument Engineering Workbench). V rámci práce se v první kapitole rozebírá grafické programovací prostředí LabVIEW a řeší se nejčastěji používané návrhové vzory užívané v tomto programovacím jazyce. Důraz je kladen na to, jakou mají strukturu, na jakých principech jsou založeny, které problémy dokáží řešit a jaká úskalí mohou přinášet. U každého vzoru je ukázka implementace v LabVIEW.

Po této kapitole následuje kapitola se specifikací zadání ze strany společnosti a rozbor testování. Zde je popsán vývoj a následné testování senzorů. V rámci kapitoly jsou obecně popisována specifika testování v oblasti Automotive. Je zde i snaha proniknout do dané problematiky a přiblížit čtenářům danou problematiku.

V praktické části se následně na základě těchto poznatků hledá vhodná softwarová architektura, která by umožňovala tyto požadavky splnit a byla by do budoucna udržitelná z pohledu jejího rozšíření a případné modifikace. Dále v praktické části práce je obsažena implementace navrhnuté architektury do vývojového prostředí LabVIEW.

Součástí práce je následně ověření daného návrhu na konkrétním senzoru a zjištění, zda daný testovací software naplňuje předdefinované požadavky a úroveň kvality.

# 2 Seznámení se s typickými úlohami v oblasti automatizace testování automobilových senzorů a požadavky na ně kladené (komunikace s přístroji, komunikace se senzorem, sběr a ukládání dat)

V automobilovém průmyslu je převážná část senzorů vyvíjena na základě zákaznických požadavků. Z tohoto důvodu je nutné prvně sepsat dokument ve kterém bude obsažena veškerá specifikace a vytváří se takzvaný RSD (Requirments Specification Document). Dá se přeložit jako dokument popisující specifické požadavky. V rámci tohoto dokumentu je popsáno, co má obsahovat a jak se má chovat HW, SW a mechanika. Na základě této specifikace se určují jednotlivé testovací metody tak, aby bylo možné ověřit, zda daný vyvíjený produkt splňuje dané požadavky. K plánování vývoje se nejčastěji využívají vývojové procesy.

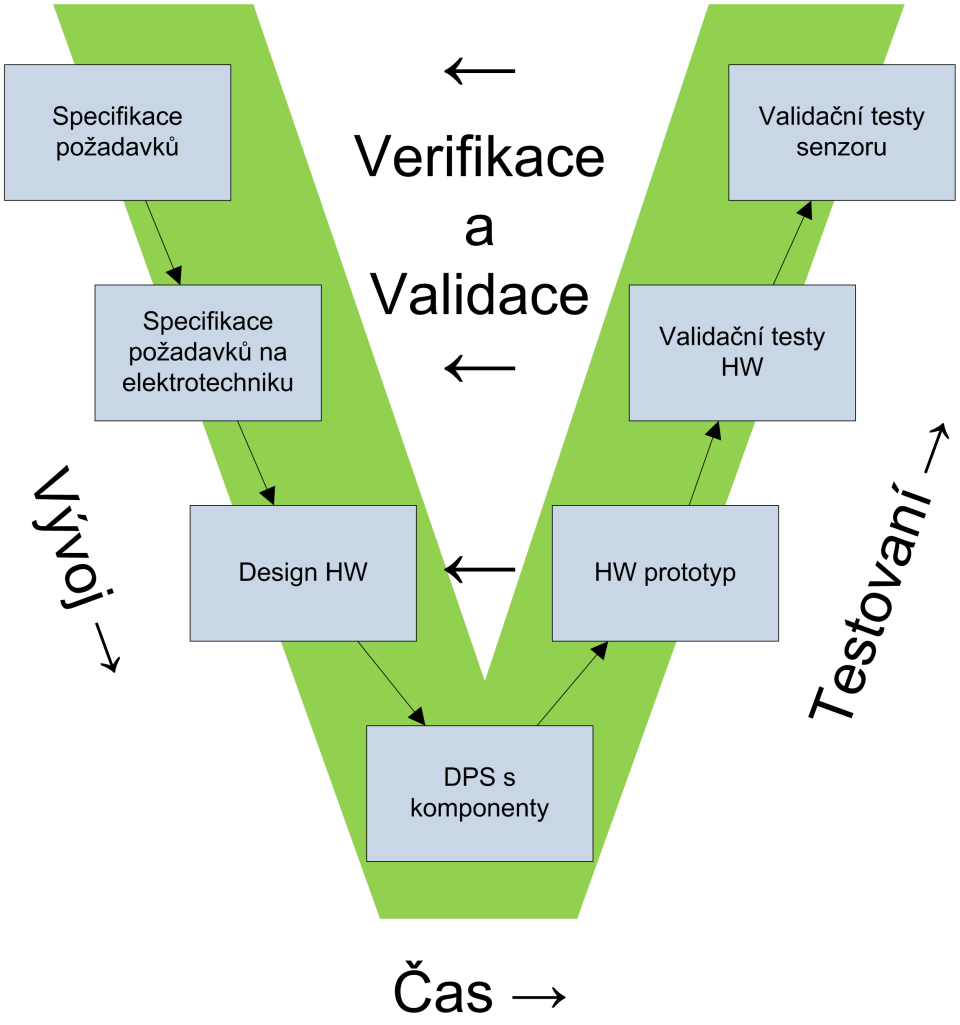
## 2.1 V-model vývojového procesu

V rámci vývoje se vývojový proces řídí takzvaným V diagramem, který uvádí shrnutí hlavních kroku, které je potřeba provést společně s daným výsledkem v dané fázi vývoje. Tím pomáhá minimalizovat rizika projektu, zaručuje kvalitu, zlepšuje komunikaci a snižuje celkové náklady na projekt.

Diagram obsahuje dvě větve, které se protínají v jednom bodu a tvoří tak tvar písmene V. V první větvi jsou specifikovány jednotlivé kroky k vytvoření fungujícího prototypu, v druhé větvi se následně daný prototyp podrobuje jednotlivým testům tak, aby bylo ověřitelné, zda skutečně splnil zadání, které bylo na začátku.

Specifikace celkového produktu je většinou daná zákazníkem, takzvaná systémová specifikace se nejprve rozdělí na dílčí specifikace pro hardware, software, konstrukci atd. V případě HW testu vznikne takzvaný ERS dokument, který specifikuje, co se od dané elektroniky požaduje. Při tvorbě ERS vzniká současně i testovací specifikace na danou komponentu tak, aby bylo možné ověřit správnost daného návrhu.

Na obrázku číslo 1 lze vidět jednotlivé kroky návrhu a následné verifikace, prvním požadavkem je sepsání kompletní specifikace, ta se většinou odvíjí od diskuze mezi zákazníkem a výrobcem. Po sepsání kompletní specifikace, která se odvíjí od požadavků zákazníka, ceny konečného výrobku a možností realizace. Ze systémové specifikace se odvodí dílčí specifikace pro mechaniku, elektroniku a software. Na základě těchto specifikací vznikají jednotlivé designy řešení, dle ERS (Electronic Reequipment Specification) vzniká design HW. Na základě tohoto designu se vytváří funkční deska plošného spoje s komponenty. Tímto je ukončena levá strana vývojového diagramu a začíná pravá testovací strana. Navrhnutý design desky plošného spoje s komponenty se následně nechá vyrobit a vzniká HW prototyp, který se podrobuje dílčím testům, které mají za cíl ověřit, zda splňují dílčí požadavky vývojového cyklu tak, aby na konci byl funkční výrobek, který splňuje specifikaci zákazníka.



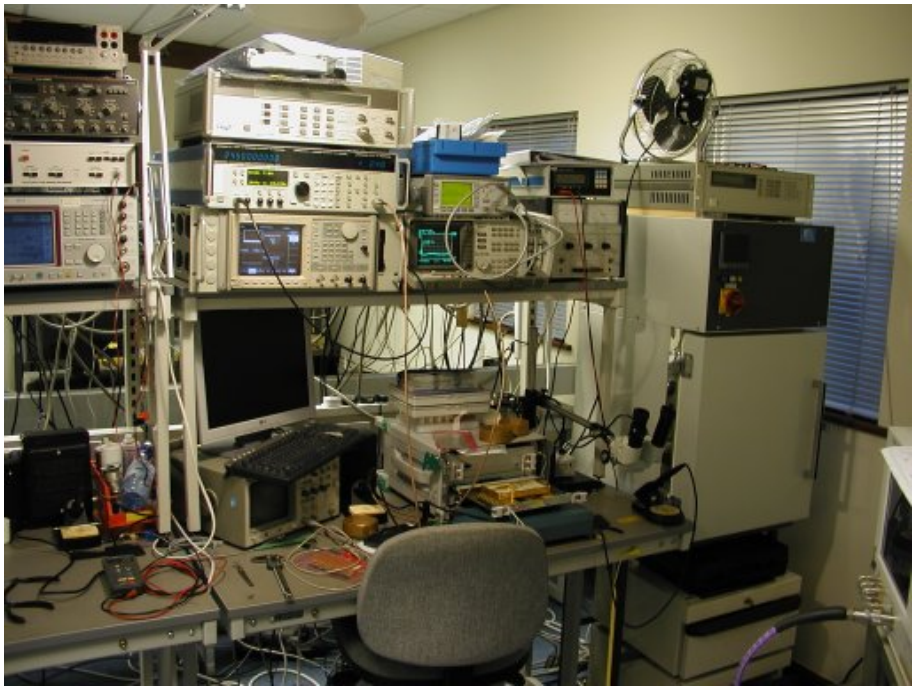

Obrázek 1: V-model vývojového cyklu

Podobný proces probíhá i při vývoji SW a mechanického designu, v určitých chvílích jsou jednotlivé procesy spojeny. Následně se daný výrobek testuje jako celek a pokud projde, je zahájena sériová výroba. Výhoda V vývojového diagramu spočívá v tom, že pokud je při testování odhalena chyba není nutné se vracet na začátek celé specifikace, ale je možné se vrátit na danou úroveň, kde testování neodpovídá plánované funkčnosti viz. obrázek číslo 1. V daném okamžiku se vždy vrací na stejnou úroveň, kdy během specifikace designu vznikala i specifikace testů. Pokud se ukáže, že to není splnitelné, je nutné jít o úroveň výš, kde bude nutné změnit specifikaci. Rovněž se může stát, že to půjde až na specifikaci zákazníka, v tomto případě se jedná se zákazníkem o možném řešení.

## 2.2 Testovací sestava

Testovací sestava je výraz pro všechna nezbytná zařízení a kabeláž, která jsou nutná pro zahájení měření. Danou testovací sestavu můžeme rozdělit na měřící přístroje, napájecí zdroje, PC a zařízení simulující okolí. Sestava se skládá z těchto kategorií:

- Do první kategorie můžeme zařadit měřící přístroje jako jsou multimetry, které slouží k měření napětí a proudu, osciloskopy na zachycení určitých dějů na desce, DAQ karty pro automatizované měření, více kanálu, případně využít Mutex (jedná se o zařízení s relátky pro přepínaní vstup na různé výstupy) a pomocí jednoho přístroje změřit více bodů na desce.
- V další kategorii jsou napěťové zdroje, pokud nejsou kalibrované, zapojují se současně s multimetrem pro ověření správné hodnoty napětí.
- PC většinou slouží na automatizovaný sběr dat z přístrojů a řízení podmínek během testu.
- Poslední kategorijí jsou zařízení pro simulaci okolí. Mezi ty můžeme zařadit simulátory termoelektrického napětí, pomocí kterého lze simulovat termočlánkové napětí a velice rychle proměřit zda daný senzor je schopný měřit ve stanoveném rozsahu. Dalším bodem jsou teplotní komory nebo olejové lázně pro testování v různých teplotách a ověření funkčnosti daného výrobku při dané teplotě.



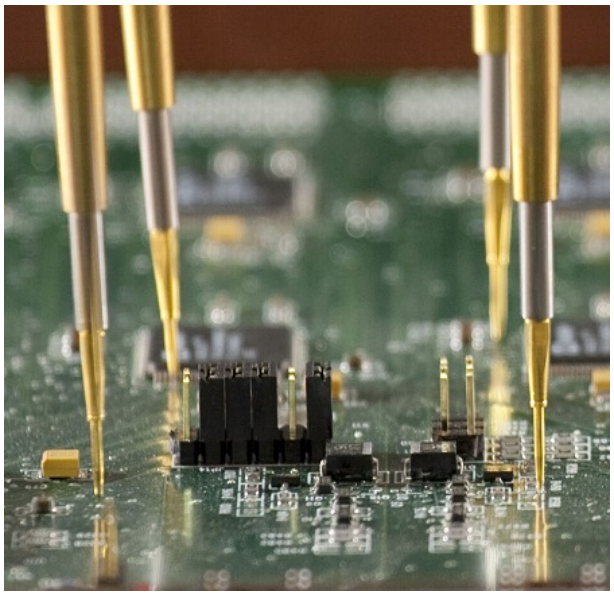
Obrázek 2: Pracoviště s testovací sestavou [1]

## 2.3 Sběr a ukládání dat

Pro komunikaci přístroje se nejčastěji využívají rozhraní GPIB nebo USB. Nad tímto rozhraním se nejčastěji využívá průmyslový standard VISA (Virtual Instrument Software Architecture). S přístrojem se následně komunikuje pomocí standardizovaných příkazů. Dalším způsobem je využití zařízení pro sběr dat NI DAQ (Data Acquisition), který slouží ke komunikaci s měřícími a řídícími kartami od NI.

Takto získaná data je následně možné nejjednodušeji ukládat do textového souboru a zpracovat v tabulkovém procesoru jako je Microsoft Office Excel, OpenOffice (LibreOffice) Calc nebo Origin od OriginLab.

Vzhledem k tomu, že se většinou testuje více bodů současně a zpravidla na více kusech, je nutné danou součástku připojit k měřícím přístrojům a napájení. K tomuto účelu se najčastěji využívá takzvaný „Needle Adapter“ neboli jehlové pole. To se skládá (jak již název napovídá) s jehel, které jsou na pružině a dosedají na testovací kontakty desky plošného spoje. Ta slouží k elektrickému připojení testované desky plošného spoje. Několik jehel následně tvoří jehlové pole, které je následně vodivě připojené většinou ke konektoru a následně k přístrojům v měřící sestavě.



Obrázek 3: Jehlové pole [2]

## 2.4 Typické prováděné elektronické testy

Jak bylo popsané u vývojového diagramu, při vývoji vzniká paralelně testovací specifikace, která má za cíl ověřit daný návrh. Většina testů je ale unifikovaných, liší se pouze od sebe kladenými parametry a designem dané desky. Tyto parametry specifikuje daná testovací specifikace. Testy můžeme rozdělit na manuální a automatické.

Manuální testy nejčastěji vyžadují práci s osciloskopem nebo detailní prozkoumaní jevu a chování bloku. Pro tyto testy z jejich povahy automatizace není výhodná, jelikož probíhají většinou na jednom měřeném zařízení, na kterém se hledá specifické chování. Do manuálních testů rovněž můžeme zařadit testy, které by sice bylo možné automatizovat, ale příprava dané měřící sestavy by zabrala více času, než ruční změření daného zařízení na stole.

Automatické testy se uplatňují u komplexních měření, která probíhají několik dní nebo může u nich být zavedena lidská chyba. Automatickým testem je lidský faktor odstraněn. Automatické měření se také objevuje tam, kde je nutné během testu měnit podmínky například, měnit teplotu teplotní komory, měnit napětí na zdroji nebo testy zaměřené na zapnutí a vypnutí senzoru. Při těchto testech se zpravidla měří více parametrů současně na více deskách současně.

Mezi typické automatické testy můžeme zařadit.

### 2.4.1 Lifetime test

Neboli akcelerovaný test, je test při kterém je výrobek testován v předem definovaných teplotách a zkoumá se zda je schopný provozu ve specifikované teplotě po určitou dobu. Testy probíhají zpravidla při vyšší teplotě, která je předem vypočtena tak, aby se dalo nasimulovat za kratší čas degradace zařízení. Ve výsledku se ověří zda daný přístroj má odpovídající životnost, aniž by bylo nutné ji testovat

### 2.4.2 HW a funkční test

Během těchto testů se ověřuje funkčnost zařízení po vyrobení. V počátku se ověřuje pouze HW funkčnost, po nafleshování příslušného firmwaru následně se ověřuje funkčnost i s komunikací atd. Tyto testy nejčastěji probíhají na NI TestStand.

### 2.4.3 Switch On-Off test

U těchto testů probíhá spínaní a zapínaní napájení na testovaném zařízení. Prvním testovacím kritériem je životnost, daný výrobek musí vydržet určitý počet cyklů zapnutí a vypnutí. Dalším kritériem je odběr proudu neboli proudová špička (Inrush Current) při sepnutí. V automobilovém průmyslu je většina senzorů napájena z řídící jednotky, ta má omezený napájecí proud, ten je definován výrobcem a je nutné tento proud nepřekročit.

### 2.4.4 Zatěžovací charakteristiky

U tohoto testu se nejčastěji zkoumá napájecí blok, zda je schopný dodávat dostatečné množství proudu. Během testu navíc dochází ke změnám vstupního napětí. Účelem těchto testů je ověřit návrh napájení a schopnost dodávat stabilní napájecí proud. U těchto testů se zpravidla testuje napěťovou rampou nebo specifikovaným úrovněmi napětí a zkoumá se odebíraný proud.

### 2.4.5 Undervolatage test

U těchto testů se rovněž zkoumá vstupní napětí na zdroji zařízení. Ve velice jemných krocích se mění napětí a zkoumá se diagnostika na komunikaci. Zda vyskakují chyby, je většinou specifikováné, aby senzor zasílal chybové zprávy, při nestandardním napětí.

### 2.4.6 Power interuption

Z pravidla bývá výrobcem deklarovano, že výrobek musí vydržet určitou dobu bez napájení, nebo určitý pokles napětí. Tyto testy se zpravidla provádějí pomocí generátoru signálu s danou frekvencí, který spíná MOSFET tranzistor pro zapnutí a vypnutí napájení zařízení. U tohoto testu se posuzuje zda je schopný provozu i s těmito výpadky.

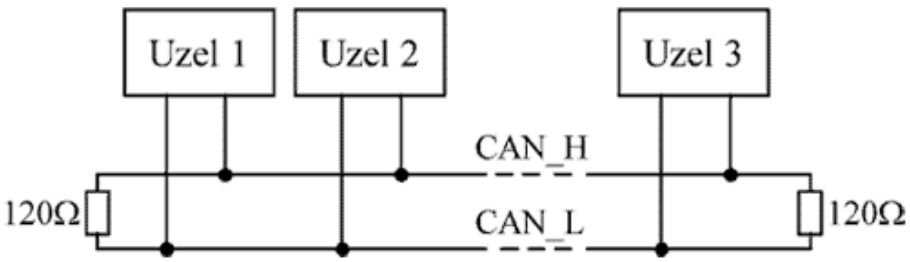
Všechny vypsané testy lze automatizovat.

## 2.5 Komunikace se senzorem

Pro komunikaci a samotné spojení s přístroji se využívají v automobilovém průmyslu různé komunikační sběrnice, mezi nejpoužívanější patří tyto: CAN, SENT, LIN a PWM.

### 2.5.1 CAN

Sběrnice CAN se nejčastěji využívá jako hlavní komunikační sběrnice v automobilu, na kterou jsou napojeny jak senzory, tak i řídící jednotky. Protokol je založený na zprávách, jedná se o robustní Multi-Master sběrnici. Sběrnice byla původně navržena pro úsporu mědi v autě, využívá proto pouze dva diferenciální vodiče CAN High a CAN Low ukončeny 60Ω nebo 120Ω terminátory. [3]

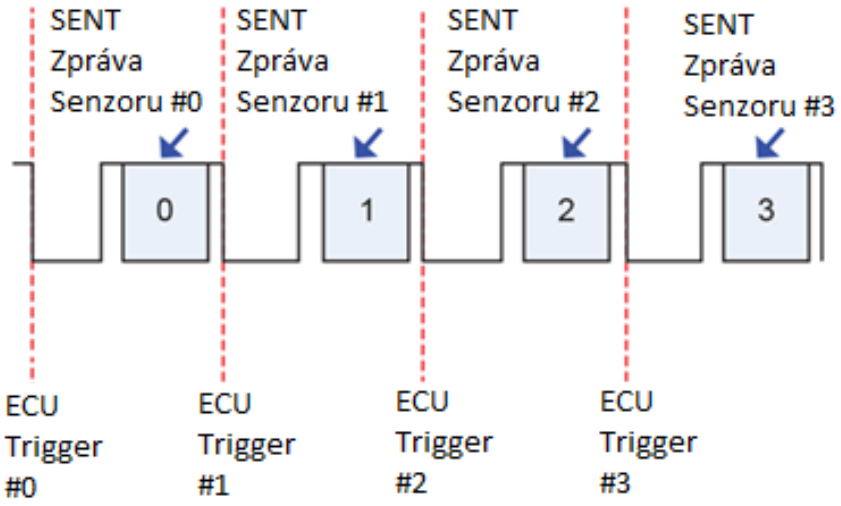



Obrázek 4: Zapojení CAN sběrnice[3]

### 2.5.2 SENT

Je jednosměrný asynchronní protokol, který pro svůj provoz vyžaduje pouze tři vodiče. Jedná se o signální vodič s TTL logikou, 5V napájecí vodič a zem. Data jsou posílána ve formě 4 bitů neboli 1 nibblu. Mezi jednotlivými intervaly je časové okno pro možnost zpracování dat na straně příjemce. SENT zpráva je dlouhá 32 bitů a obsahuje 24 bitů dat a 8 bitů kontrolního

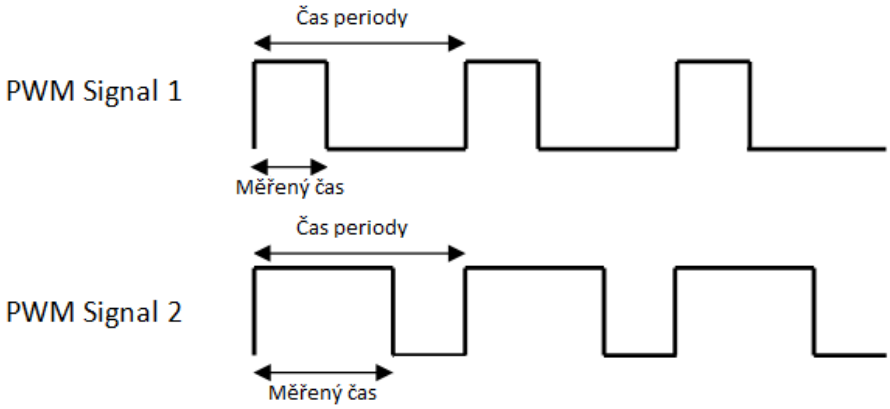
součtu. Sběrnice není robustní jako CAN a využívá se nejčastěji pro přímou komunikaci senzoru s nadřazenou jednotkou. [4]




Obrázek 5: Segmenty SENT sběrnice[4]

### 2.5.3 PWM

Jedná se o pulzně šířkovou modulaci, která se nejčastěji využívá k číslicovému řízení výkonu. využívá se též i pro komunikaci u jednodušších senzorů, kde slouží tato sběrnice pro zasílaní hodnoty, na straně příjemce se měří doba pulzu, která je následně přepočtena na číselnou hodnotu, která může odpovídat teplotě, tlaku atd. Perioda vůči které se vztahuje střída je pevně daná specifikací. [5]



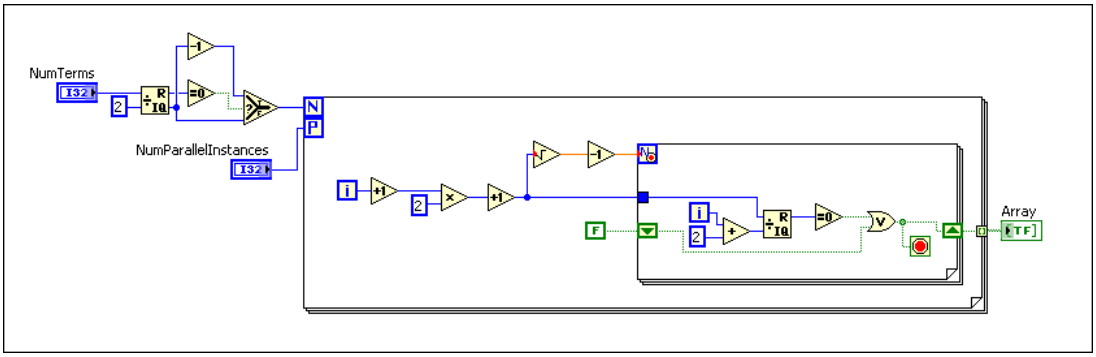

Obrázek 6: Dekódovaní PWM

# 3 Úvod do problematiky vytváření aplikací v LabVIEW, možné přístupy k jejich tvorbě

## 3.1 Obecně o LabVIEW

LabVIEW je grafické programovací prostředí, které může být využito pro vývoj aplikací s grafickým uživatelským prostředím neboli GUI z anglického (Graphical User Interface). Původně bylo vyvinuto pro Apple Macinstosh z roku 1986. Ve vývojovém prostředí se používá grafický programovací jazyk se jménem „G“ (neplést s G-kódem, který je určen pro NC a CNC stroje). Pokud technicky mluvíme o LabVIEW, myslíme tím samotné grafické vývojové prostředí a „G“ samotný grafický programovací jazyk, který dané prostředí využívá. Ale vzhledem k tomu, že samotný grafický programovací jazyk „G“ je využit pouze ve vývojovém prostředí LabVIEW, tak samotným termínem LabVIEW se obecně myslí oboje, jak vývojové prostředí, tak samotný grafický programovací jazyk.

Název LabVIEW je zkratka anglických slov Laboratory Virtual Instruments Engineering Workbench, což se dá volně přeložit jako laboratorní virtuální přístrojová technika. Tento nástroj je široce využíván zejména inženýry, ale i vědci a studenty k rychlému vývoji aplikací pro měřící, testovací a řídící účely. Vývojové prostředí a samotný programovací jazyk umožňuje stejné dovednosti jako ostatní programovací jazyky, je tedy v něm možné pracovat s proměnnými v různých datových typech, poli, řetězci, používat smyčky, podmínky, struktury atd. LabVIEW je událostmi řízený a objektově orientovaný jazyk. Kompilace probíhá na pozadí při vytváření kódu aplikace. [6] [7] [8]




Obrázek 7: Ukázka LabVIEW aplikace

## 3.2 Princip grafického programování a dataflow

Základní entita kód v LabVIEW, takzvané VI je rozděleno na dva celky, na blokové schéma programu a grafické uživatelské prostředí. Čelní panel je sestaven z ovládacích prvků a indikátorů. Tyto prvky jsou obsaženy i v blokovém diagramu jako jsou terminály a je na ně možné přivést nebo vyvést data. I když má LabVIEW mnoho společného z logiky věci s klasickým řádkovým

programováním, jsou tu jistá specifika. Samotný princip fungování vykonávaní kódu v grafickém jazyce „G“ je dostupnost dat. Pokud má funkce na všech vstupech data tak se provede. Není zde tedy zajištěna posloupnost, jako u řádkového programování, kde je kód vykonáván po řádcích. Zde je posloupnost zajištěna pouze na závislosti dat na vstupech jednotlivých funkcí. Pokud žádná datová závislost není, tak výpočty se vykonávají na sobě nezávisle, tedy paralelně, a není zaručeno, který kus kódu proběhne dřív. Takovému běhu se v terminologii LabVIEW říká DataFlow. Propojení jednotlivých funkcí probíhá skrz graficky znázorněné vodiče. Multi-procesní a multi-vláknové operace jsou implementovány vnitřním plánovačem procesů, který provádí dané operace nad daným operačním systémem. [6] [7]

## 3.3 Virtuální instrumentace

Jednotlivé programy se nazývají virtuálními instrumenty a většinou se užívá jen jejich zkratka v podobě VI. Ty se dají následně využít jako podprogram neboli subrutina (z ang. Subroutine). Daná subrutina se následně nazývá sub-VI. [6] [7]

## 3.4 Návrhové vzory

V softwarovém inženýrství se jedná o obecné opakovatelně použitelné řešení běžně se vyskytujícího problému. Daný problém má mnoho různých způsobu řešení a designové vzory formulují nejlepší možné řešení. Nejedná se o kompletní kód, který by bylo možné rovnou použít, ale o popis nebo šablonu, jak daný kus problému řešit. V objektově orientovaných návrhových vzorech se zobrazují vztahy a interakce mezi třídami nebo objekty. Mezi návrhové vzory nepatří algoritmy, protože řeší daný problém konkrétně, nikoliv obecně. Užití návrhových vzorů vede k urychlení procesu vývoje, zároveň při opětovném použití vede ke zmírnění chyb programátora, z důvodu lepší čitelnosti kódu a při správně zvoleném návrhovém vzoru se vyhne případnému přepisování kódu z důvodu chyb v návrhu.

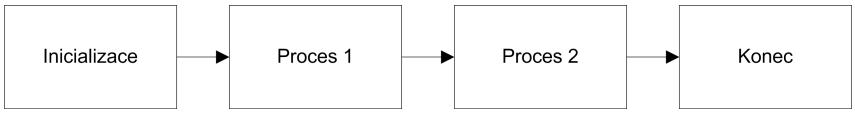
- Návrhové vzory můžeme rozdělit na:
  - Creational Patterns (vytvářející) – řeší problémy související s vytvářením objektů v systému. Snahou těchto návrhových vzorů je popsat postup výběru třídy nového objektu a zajištění správného počtu těchto objektů. Většinou se jedná o dynamická rozhodnutí učiněná za běhu programu.
  - Structural Patterns (strukturální) - představují skupinu návrhových vzorů zaměřujících se na možnosti uspořádání jednotlivých tříd nebo komponent v systému. Snahou je zpřehlednit systém a využít možností strukturalizace kódu.
  - Behavioral Patterns (chování) - se zajímají o chování systému. Mohou být založeny na třídách nebo objektech. U tříd využívají při návrhu řešení především principu dědičnosti. V druhém přístupu je řešena spolupráce mezi objekty a skupinami objektů, která zajišťuje dosažení požadovaného výsledku. [9]

- Níže jsou popsány nejčastější návrhové vzory, které se využívají v LabVIEW aplikacích.
  - State Machine
  - Event-Driven User Interface
  - Master-Slave
  - Producer-Consumer
  - Queued State Machine with Event-Driven Producer-Consumer

### 3.4.1 Stavový automat (State Machine)

Stavový automat je jednou ze základních architektur, které často vývojáři používají k rychlému vytváření aplikací. Stavová struktura může být použita k složitým rozhodovacím algoritmům, které se dají reprezentovat stavovými diagramy. Přesněji řečeno stavový automat implementuje vazby mezi jednotlivými stavy, kdy každý požadavek má svoji specifickou akci a algoritmus pro každý stav v diagramu. Stavový automat můžeme dle algoritmu rozhodnutí, jaký stav bude následovat, rozdělit do následujících kategorii:

- Statická sekvence – jednotlivé stavy mají pevné pořadí
- Dynamická sekvence – pořadí mezi stavy se řídí na základě nějaké podmínky

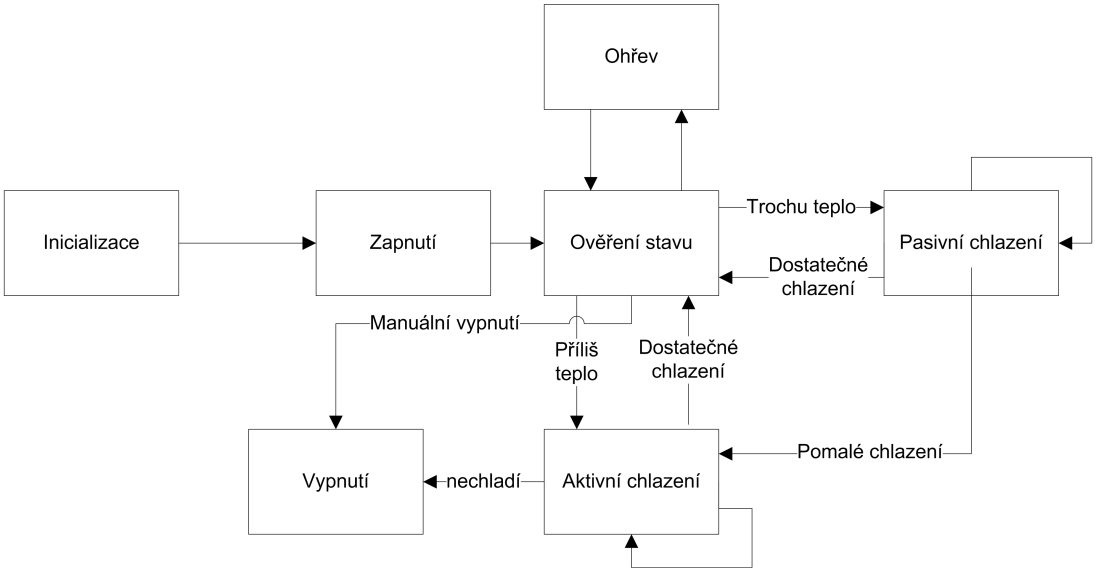



Obrázek 8: Statická sekvence stavového automatu

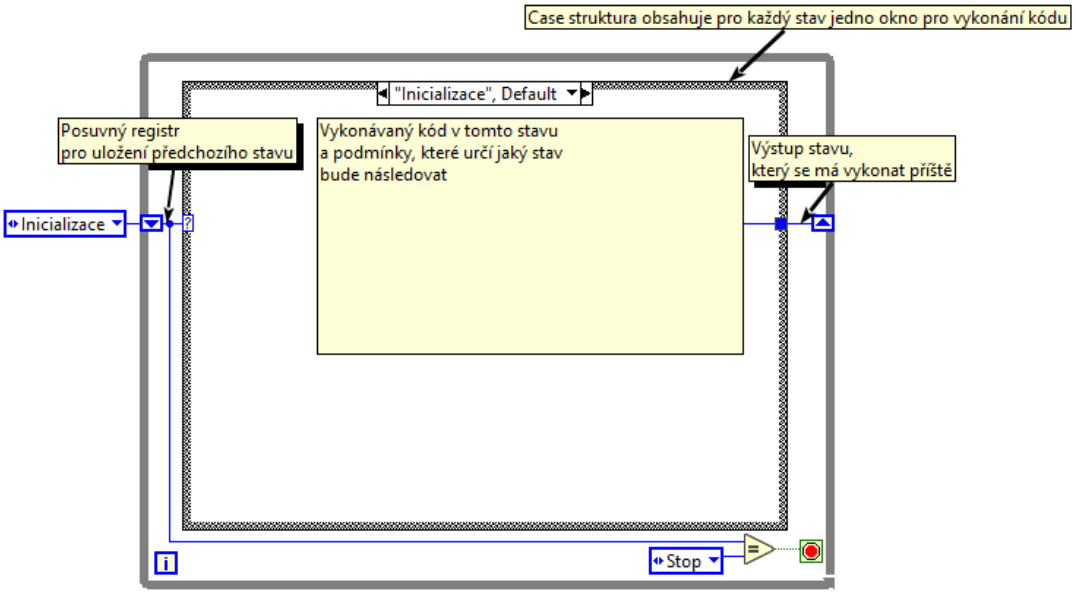
Mnoho aplikací vyžaduje stav „inicializace“, po kterém následuje výchozí stav, kde následně dojde k rozhodnutí dalšího stavu. Prováděné akce přepínaní stavů mohou záviset na předchozích a současných vstupech i stavech. Stav „vypnout“ pak může být použit k provedení čistících akcí a následnému ukončení programu.

Stavové stroje jsou nejčastěji používány při programování GUI neboli uživatelských rozhraní. Stavový stroj sleduje uživatelské akce a na základě nich se rozhoduje. Existuje ještě varianta stavového stroje, takzvaný Queued Message Handler. Jedná se o sofistikovanější verzi stavového stroje a nabízí větší flexibilitu a zároveň větší složitost. [10] [11]

- Nevýhody návrhového vzoru:
  - Každý Case je stav
  - Momentální stav, znemožňuje vykonávání kódu následujícího stavu dokud neskončí
  - Využití enumerátoru v posuvném registru pro definici jednotlivých stavů




Obrázek 9: Dynamická sekvence stavového automatu na příkladu chladiče

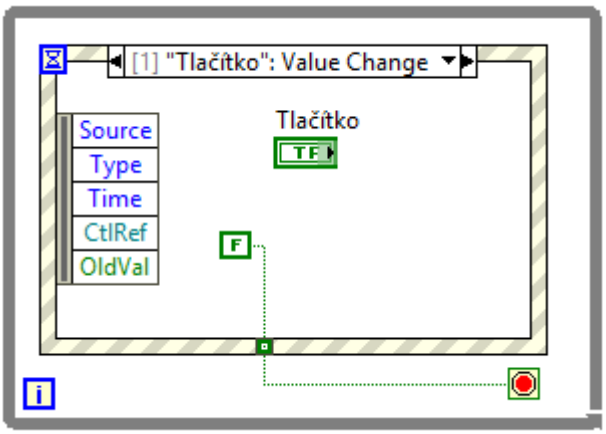



Obrázek 10: Implementace stavového automatu v LabVIEW

### 3.4.2 Událostně řízená obsluha uživatelského prostředí (Event-Driven User Interface)

Událostně řízena obsluha uživatelského rozhraní se používá, když je potřeba se dotazovat na uživatelské akce, nepromeškat uživatelskou událost a zároveň nebrzdit průběh programu příliš rychlým dotazováním.

Základním požadavkem pro událostně řízenou strukturu je použití Event Structure, událostně řízené struktury, která dokáže odchytávat notifikace z čelního panelu zajišťujícího GUI




Obrázek 11: Implementace Event-driven user interface v LabVIEW

aplikace.

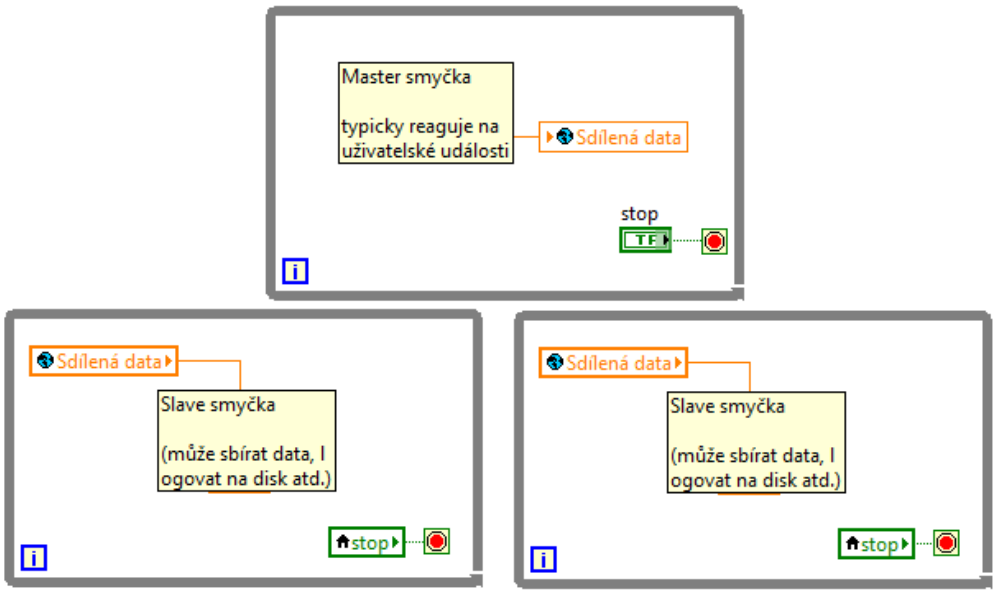
1. Operační systém pošle zprávu skrz broadcast na základě systémové události (kliknutí myši nebo klávesnice) do aplikací.
2. Event Structure zachytí a zaznamená událost a vykoná události přiřazený kód.
3. Event Structure vrací informaci ohledně proběhnutí události v Case
4. Event Structure přiřadí událost do fronty, pokud je zrovna systém zaneprázdněný.

Událostně řízena struktura je umístěna ve smyčce while, smyčka je následně brzděná do zaznamenání příslušné události nebo vypršení časového limitu. Kromě zachycení událostí interakce z čelního panelu lze pomocí struktury zachytávat dynamické události implementované v programu. [11]

### 3.4.3 Struktura pán a otrok (Master-Slave)

Tento vzor je výhodný při vytváření víceúlohových aplikací (multitaskingu). Poskytuje nám více modulární přístup k vývoji aplikací, protože není založena na jedné centrální smyčce pro vykonávaní kódu, což zároveň poskytuje kontrolu nad časováním. V LabVIEW se každá paralelní smyčka zpracovává jako samostatný proces nebo vlákno. Vlákno se dá definovat jako část programu, která se může vykonávat nezávisle na ostatních částech aplikace. Pokud se v aplikaci nepoužívá více vláknová funkcionalita, tak se daná aplikace chová jako jedno vlákno. Pokud ale rozdělíme aplikaci na více vláken, tak každé vlákno nesdílí stejnou výpočetní dobu. To ale vyžaduje kontrolu nad tím, jak bude aplikace časována neboli synchronizována.

Standardním přístupem pro tvorbu návrhového vzoru Master-Slave je využití dvou paralelních smyček (Slave), které jsou podřízené hlavní smyčce (Master). Ta provádí obsluhu uživatelského rozhraní neboli GUI. Při komunikaci s podřízenými smyčkami hlavní smyčka zapisuje do




Obrázek 12: Implementace Master a Slave v LabVIEW

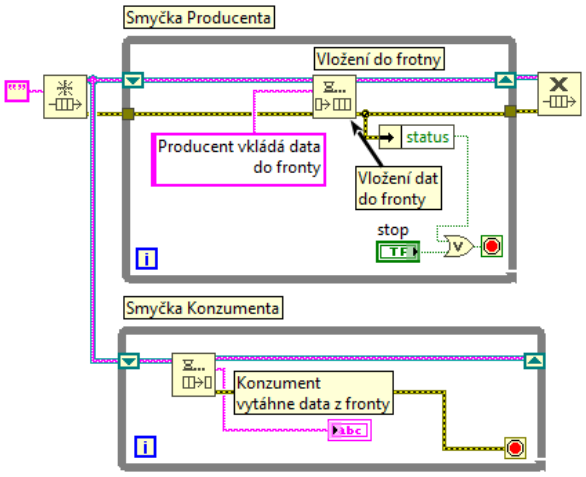
lokálních proměnných, a tím zajistí že každý podřízený proces neovlivní druhý proces. Tento způsob ale není oboustraně bezpečný z důvodu možnosti přepisu nevyčtené hodnoty. Dále je zajištěno, že prodlení hlavní smyčky, například z důvodu vyvolání dialogu v GUI, nebude mít důsledky na zpoždění procesů v podřízených smyčkách, pokud běží v jiných VI. [12] [11]

### 3.4.4 Producent a konzument (Producer-Consumer)

Tento vzor má základ ve vzoru Master-Slave a snaží se zdokonalit sdílení dat mezi smyčkami. Stejně jako u Master-Slave vzoru, i u producenta-konzumenta dochází k oddělení procesů, které tak mohou plnit nebo zpracovávat data různou rychlostí. Paralelní smyčky jsou rozděleny do dvou kategorií na ty, které data produkují (producent) a na ty, která data spotřebovávají (konzument). Pro datovou komunikaci mezi smyčkami se většinou využívají datové fronty, ty nabízejí výhodu vyrovnávaní datového toku mezi smyčkami producenta a konzumenta.

Vzor je hojně využíván, když je potřeba získávat více dat než je možné v ten okamžik zpracovat. Předpokládá se, že je potřeba napsat aplikaci která bude přijímat data a následně je bude zpracovávat v pořadí v jakém byla přijata. Pokud je rychlost generování dat mnohem vyšší než samotné zpracování dat, tak je Producent konzument nejlepší možný návrhový vzor.

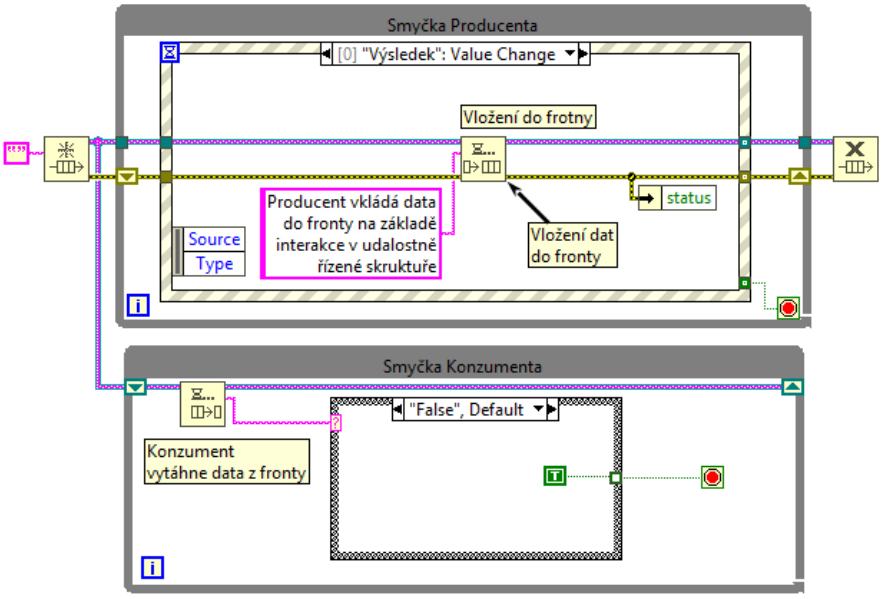
Stejně jako návrhový vzor Master-Slave, tak se i návrh konzument-producent skládá z paralelních smyček, komunikace mezi smyčkami následně probíhá nejčastěji skrz frontu. [11] [13]




Obrázek 13: Implementace Producenta a konzumenta v LabVIEW

### 3.4.5 Frontový stavový automat s událostně řízeným producentem a konzumentem (Queued State Machine with Event-Driven Producer-Consumer)

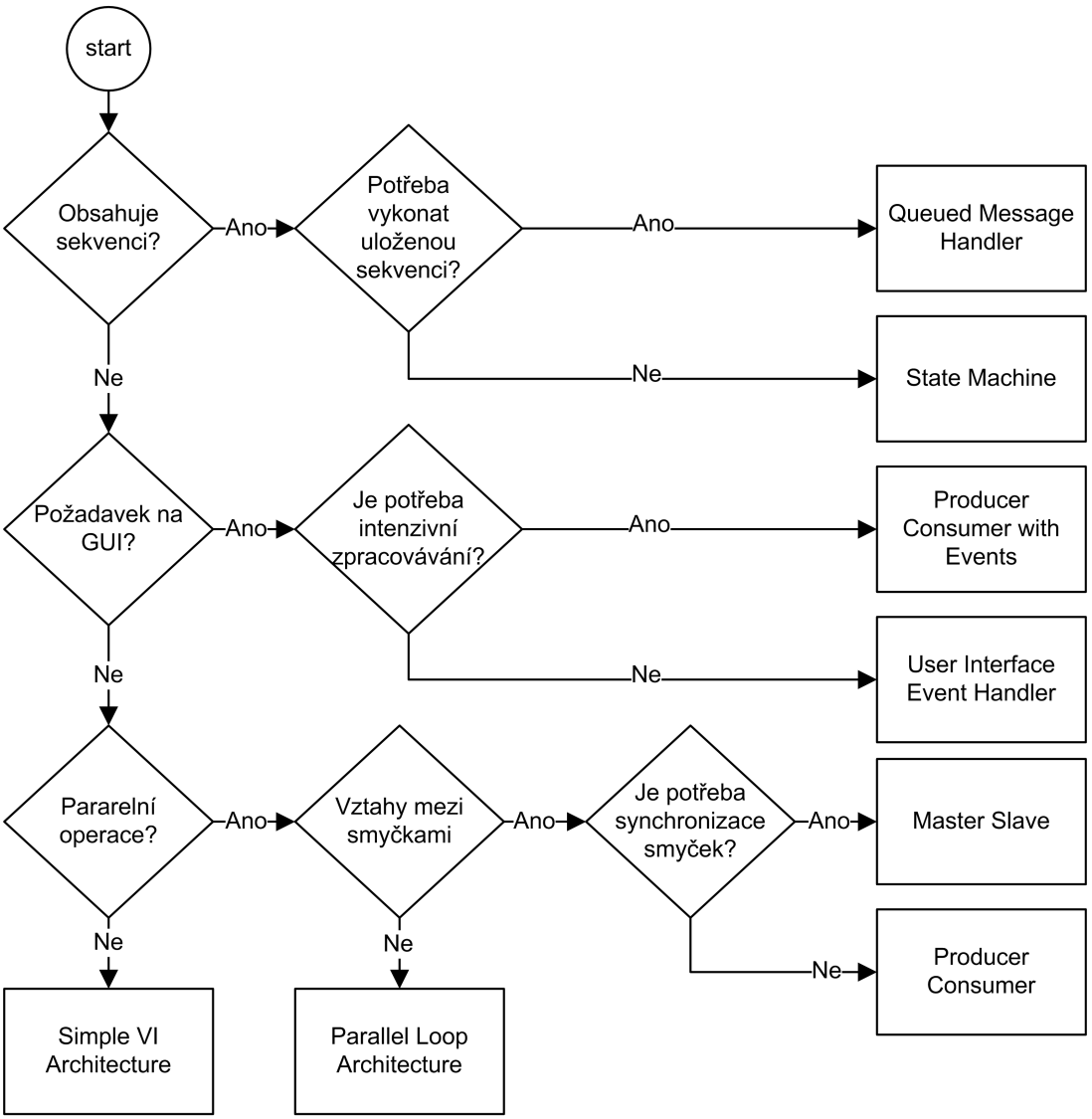
Jedná se o kombinaci stavového automatu, událostně řízené obsluhy uživatelského prostředí a producenta konzumenta. Obsluha je zde zpracovávána v jedné smyčce pomocí událostně řízené struktury a je producentem požadavků na druhou smyčku konzumenta, která konzumuje přijaté zprávy a na základě stavového automatu vykonává dané události. Jedná se o nejvíce rozšířený návrhový vzor pro tvorbu rozsáhlejších aplikací v NI LabVIEW.




Obrázek 14: Implementace Queued State Machine with Event-Driven Producer-Consumer v LabVIEW

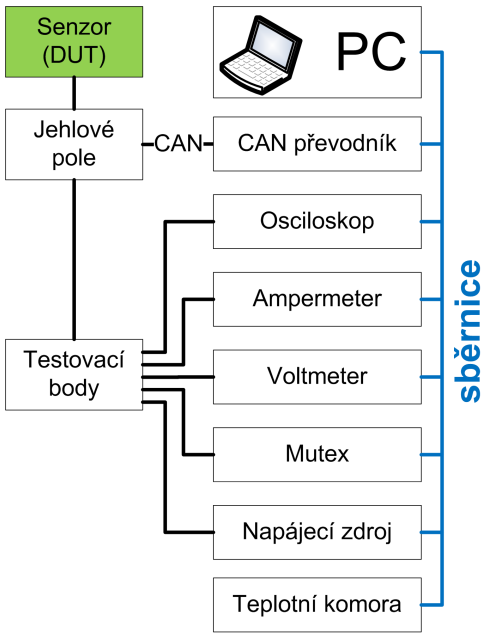
## 3.5 Výběr vhodného návrhového vzoru

Uvedený diagram popisuje rozcestník k vybrání nejvhodnějšího návrhového vzoru, kromě výše uvedených obsahuje i neuvedené méně používané návrhové vzory v labVIEW. Je patrné že, vzor State-Machine se hodí pro sekvenční události nebo zpracování dat a zároveň je důležité si ponechat možnost programovatelného rozhodování. U návrhového vzoru producent konzument je možné vykonávaní dvou procesů v jeden čas a zároveň zachovat, aby jeden proces nezpomaloval druhý. [11] [14]




Obrázek 15: Diagram popisující výběr vhodného návrhového vzoru

# 4 Návrh modulární architektury SW pro oblast automatického testování automobilových senzorů




Obrázek 16: Typický příklad měřící sestavy

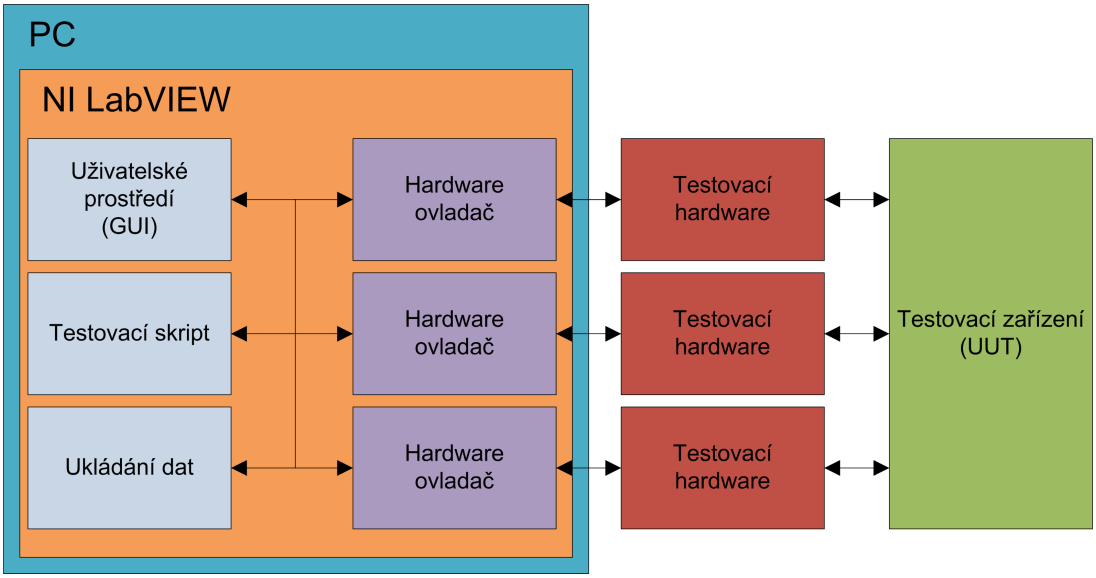
Jak bylo zmíněno v předchozí kapitole, typická měřící sestava může vypadat jako na obrázku číslo 16. Je v ní obsaženo několik přístrojů, které komunikují s PC. Je u nich potřeba zajistit součinnost a mít možnost porovnat jednotlivé hodnoty a jejich závislost mezi sebou.

## 4.1 Požadavky na architekturu

Hlavní požadavky na danou architekturu byly:

- Modularita
- Synchronizace měření z různých přístrojů a zařízení
- Zrychlení vývoje testeru pro danou měřící konfiguraci
- Snížení nákladů na vývoj v člověkohodinách
- Znovuvyužití modulů, pro jinou konfiguraci měřící sestavy projektu

Tyto požadavky vzešly z testovacího oddělení pro HW, kde vznikl požadavek na stavebnicové řešení s minimálními úpravou zdrojového kódu konkrétního testeru. Je běžné, že v rámci HW testů se opakují úlohy jen s mírnou změnou v konfiguraci. Například totožný senzor používá místo CAN sběrnice SENT sběrnici. Tato nepatrná změna často vedla k napsání nového testeru, jelikož nebylo možné snadno přestavět daný koncept.



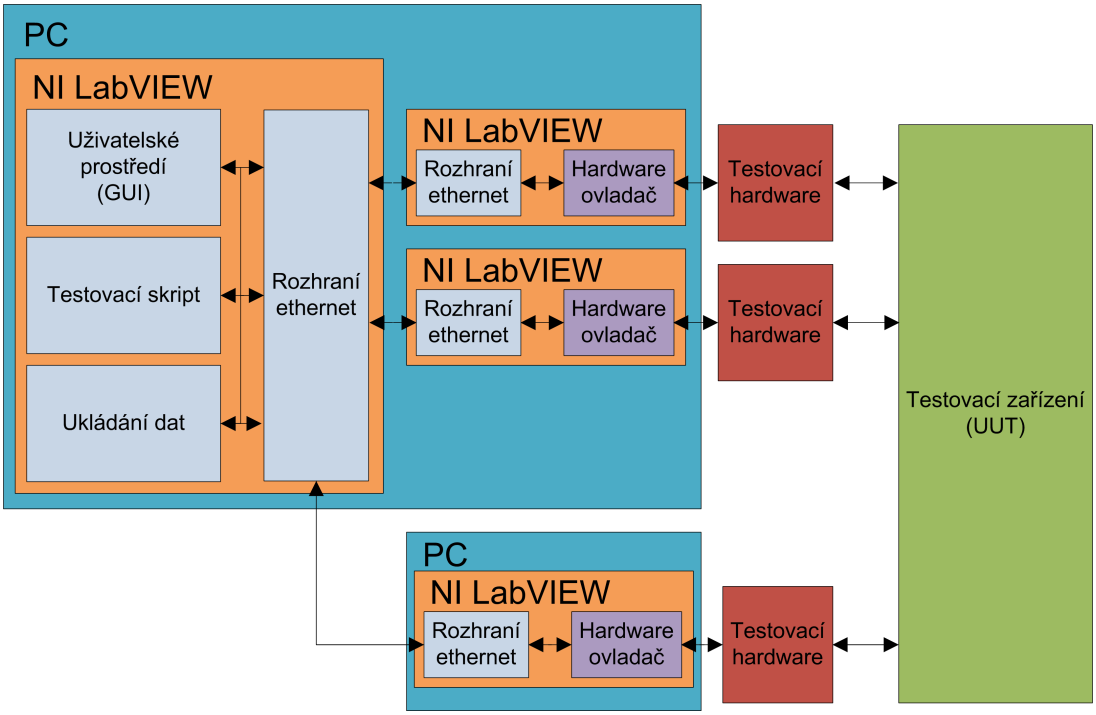

Obrázek 17: Možné řešení

Další myšlenkou v době návrhu bylo vytvořit návrh zahrnující v sobě Ethernet a přes lokální síť použít více počítačů pro jedno konkrétní měření. Tento návrh byl ale později zamítnut z důvodu složitosti daného řešení a robustnosti. V důsledku překladu vrstev TCP/IP protokolu a latence by nemuselo být zaručeno zaslaní daného měření včas. Musela by se ošetřit synchronizace času mezi moduly tak, aby bylo možné triggrovat na předem daný čas, který by nemusel být zaručen. Tak aby každý modul do tohoto času obdržel zprávu a mohl ji zpracovat viz. obrázek č. 18. Z konzultace vyplynulo, že by dané řešení nebylo ani plně využito. Většina zařízení má USB rozhraní a plně dostačuje využití jednoho PC a jeho možnosti konektivity lze rozšířit pomocí USB HUB modulu.

## 4.2 Možná řešení SW architektury

Prvotní myšlenkou by bylo oddělit jednotlivé sekce kódu do bloků, které by bylo možné nezávisle na sobě používat, zaměňovat a popřípadě doplňovat moduly. Například již zmíněný příklad s CAN a SENT sběrnicí. Pro nový senzor by stačilo pouze prohodit tyto dva bloky. Princip by spočíval v užití producenta a konzumenta se správným rozhraním pro komunikaci mezi moduly, kdy jednotlivé bloky provádějí nezávislá měření ve svých smyčkách a nasbíraná data zasílají do testovacího skriptu, kde proběhne zpracování a následné uložení. Kromě toho by byla obsažena nezávislost běhu na uživatelském rozhraní. Tento koncept by byl plně vyhovující pro jednoúčelový tester, kde v případě výměny komponenty by stačilo přepsat ovladač. Ukázalo se, že tento přístup není modulární.

Dalším krokem tedy bylo zahrnout testovací skript do modulu tak, aby byl nezávislý na zařízení a poskytoval unifikovaný výstup pro ukládání dat pro výsledné vyhodnocení. Mohl nastat případ, kdy stejný senzor pouze SW úpravou měl jiný datový rámec CAN zprávy.



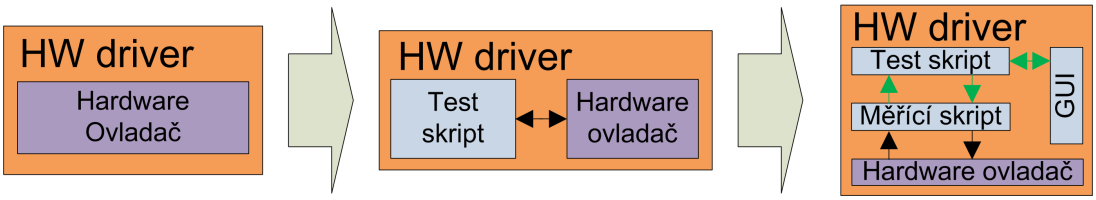

Obrázek 18: Možné řešení s Ethernetem

## 4.3 Dopracování k zvolenému řešení

V průběhu návrhu architektury se došlo do mnoha slepých uliček. V první zvažované variantě bylo uvažováno o jedné společné frontě na předávání příkazů, tak i na vracení dat. Tato myšlenka se později ukázala jako nevhodná z důvodu nemožnosti spustit všechny měřící elementy v jeden čas. Další problém by nastal při opoždění triggeru z důvodu vracení dat z měřící struktury. Bylo tedy nutné oddělit komunikaci která zajišťovala výměnu dat od komunikace pro příkazy.

### 4.3.1 Vývoj HW driveru

Jak rostla s postupem času složitost programu, bylo nutné HW postupně modifikovat. V počátku byl HW driver koncipován jako pouhé rozhraní pro komunikaci s přístroji a počítalo se zpracováním dat v MAINu. Tento koncept byl posléze vyměněn za zpracování dat přímo v HW driveru v takzvaném testovacím skriptu. Tento koncept nebyl také plně ideální, a proto bylo ve finální verzi přidáno uživatelské rozhraní pro možnost ladění HW driveru nezávisle na MAINu. Dále přibyl měřící skript reagující na příkazy z MAINu pro provedení měření. Testovací skript byl ponechán na zpracování a zasílání dat do MAINu. Posloupnost je znázorněna na obrázku číslo 19.




Obrázek 19: Postup vývoje HW driveru

## 4.4 Koncept zvoleného řešení

Aby byla zaručena bezproblémová funkčnost a možnost testování, je možné HW drivery spustit samostatně. V základu je v HW driveru obsažena pouze šablona se všemi nezbytnými obslužnými generickými (obecnými) rutinami. Na programátorovi testu následně zbývá pouze doprogramovat obsluhu přístroje, tím je myšlena konfigurace komunikace VISA, DAQ (nebo jiného rozhraní pro sběr dat), napsání skriptu pro následné zpracování dat (rozkódování přijatého rámce zprávy, přepočet jednotek atd.).

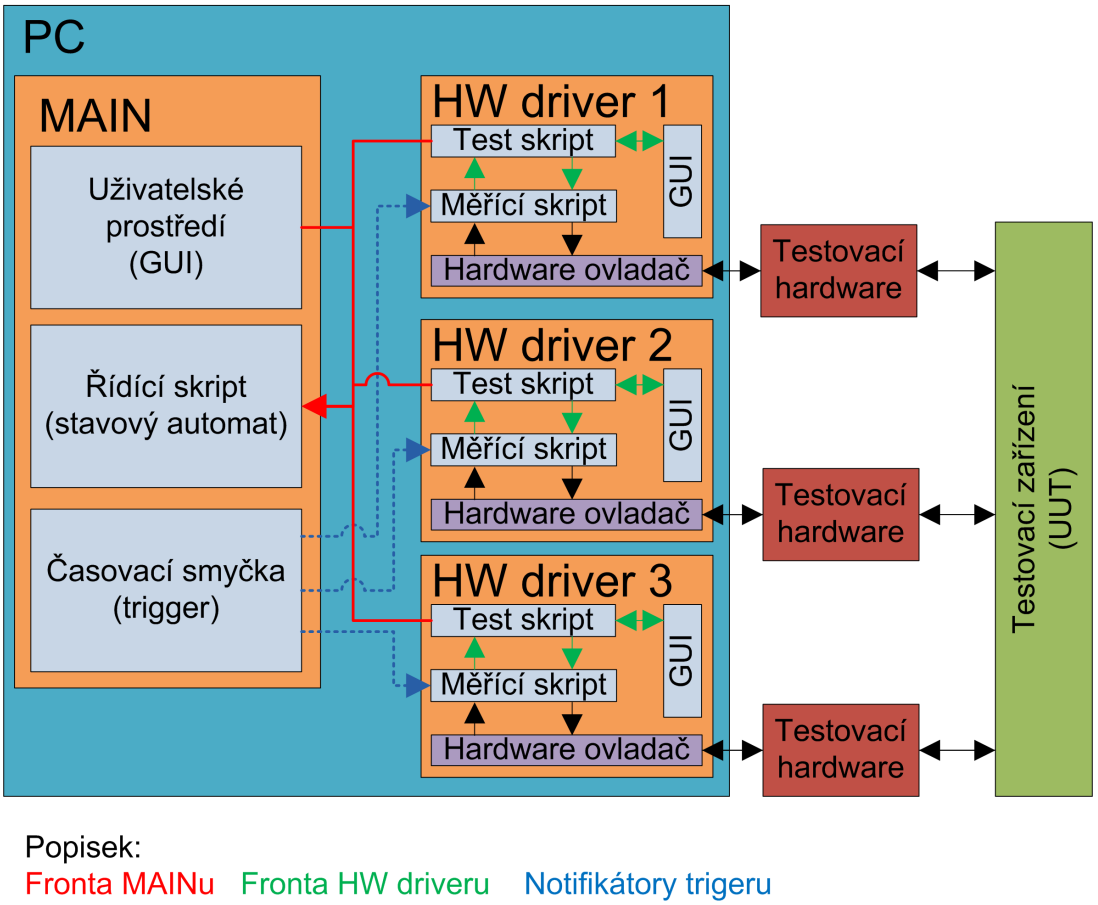
Následně takto modifikovanou šablonu stačí pouze uložit a samostatně spustit jako nový HW driver. Tím má programátor možnost ladit jenom dané VI HW driveru, aniž by musel řešit synchronizaci s ostatními VI. Programátorovi rovněž odpadá řešit ukládání dat, komunikaci mezi moduly nebo časování a posloupnost jednotlivých modulů. Podmínkou je pouze, aby daný HW driver měl unifikovaný vstup a výstup dle znázorněné architektury. Unifikace vstupů a výstupu daného HW driveru spočívá v odesílání dat v takovém formátu, který již bude obsažen v logu a schopnost reagovat na generické vstupy v podobě textových příkazů.

Po napsání jednotlivých HW driverů následně stačí skrz hlavní řídící VI, tedy MAIN spustit a vzdáleně volat skrz příkaz „TRIGGER“ strukturu v HW driveru, pro získání naměřených dat. Naměřená data jsou následně již v zpracované podobě převzata a uložena do logu. Z následného logu se pak dají data manuálně zpracovat například v tabulkovém procesoru jako je Microsoft Excel.

## 4.5 Zvolené řešení

Zvolené řešení bylo navrženo následovně: z původních modelů zůstal samostatný hlavní program neboli MAIN obsahující obsluhu uživatelského prostředí, řídící skript obsahující stavový automat a časovací smyčka, zajištující volání jednotlivých HW driverů. Architektura je znazorněna na obrázku 20.

Časovací smyčka MAINu pro volání HW driverů je koncipován pouze na dotazování jednotlivých testovacích skriptů v HW driverů přes notifikátory. Notifkitátor byl oproti frontě zvolen z několika důvodů. Používá se pouze na generické předem dané příkazy, není zapotřebí mít buffer (FIFO) paměť. Vzhledem k absenci paměti a nutné režie je z principu rychlejší z pohledu latence. Tímto způsobem je zajištěno, že příkaz pro měření proběhne co možná v nejbližším časovém okamžiku, ideálně najednou. Je rovněž do jisté míry zaručeno, že měření začne ve stejný




Obrázek 20: Zvolené řešení

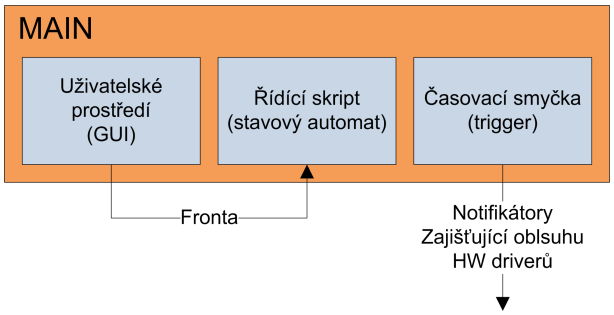
čas. Časovací smyčka MAINu obsahuje plánovač a umožňuje volání jednotlivých HW driverů v přesných intervalech, tedy je možné zajistit volání HW driverů v rozdílných periodách, jedinou podmínkou je aby daná perioda byla celočíselným násobkem periody hlavního časovače.

Po následném příkazu triggeru z MAINu do HW driveru se zašlou skrz notifikátor naměřená data zpět do fronty MAINu pro uložení do společného souboru. Výhodou tohoto řešení je, že HW driver obsahuje jak nastavení samotného měřícího hardware, tak i testovací skript, který zpracuje data do odpovídající podoby. Při změně je tedy možné vyměnit tento blok celý. V případě změny protokolu nebo kritérií je změněn pouze testovací skript. U změny HW je změna u ovladače jako je VISA nebo DAQ řešena v Měřícím skriptu. Koncept počítá se spuštěním těchto aplikací jako nezávislých procesů, tedy v samostatných vláknech. Procesy se tedy navzájem neruší a mohou být plně nezávislé. Architektůra je znázorněna na obrázku 20.

## 4.6 Hlavní řídící VI (MAIN) a HW drivery

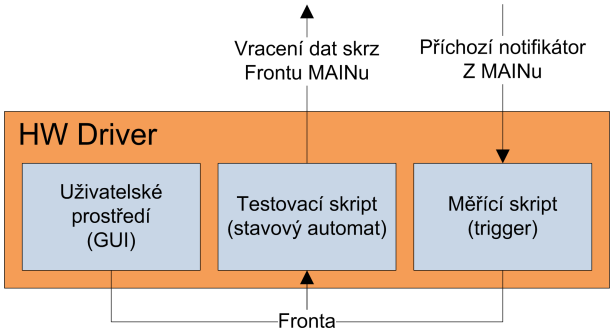
Hlavní řídící VI (MAIN) se skládá ze tří částí. V první je obsažena obsluha grafického prostředí, v němž je zajištěna uživatelská interakce a konfigurace programu. Jednotlivé příkazy se následně

předávají pomocí vnitřní fronty do druhé části programu, kde je obsažen stavový automat. Stavový automat je řízen příkazy obsaženými ve frontě. V této části se vykonává veškerý kód. Poslední částí je pak časovací smyčka pro spuštění, která zajišťuje v pravidelných intervalech zasílání příkazů pro měření jednotlivým HW driverů.




Obrázek 21: Struktura MAINu

Struktura HW driveru je řešena obdobně. I zde je obsažena jedna společná interní fronta pro řízení stavového automatu jak z GUI, tak i z Měřícího skriptu kde je obsažena měřící struktura, čekající na trigger z MAINu pro spuštění měření. Změřená data jsou následně předány přes interní frontu do testovacího skriptu na zpracování a následně předány do fronty MAINu kde dojde k uložení.

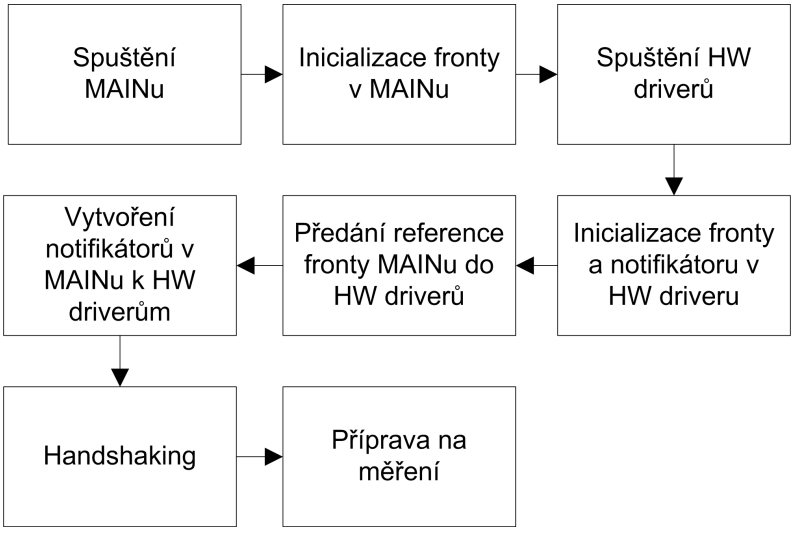



Obrázek 22: Struktura HW driveru

Implementace je ukázana v kapitole: „Implementace SW dle návrhu ve vývojovém prostředí LabVIEW“.

## 4.7 Spuštění programu a komunikace mezi hlavním VI (MAINem) a HW drivery

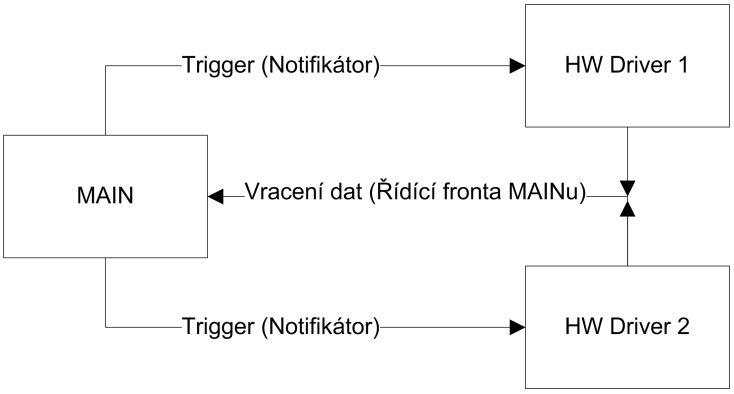
Po spuštění MAINu proběhne spuštění HW driverů skrz zadanou referenci cesty. Dané spuštění proběhne na pozadí v samostatných procesech. Po spuštění je komunikace mezi MAINem a HW driverem zajištěna notifikátorem. Vzhledem k tomu, že název notifikátoru je odvozen z cesty adresářové struktury přidáním řetězce „_noti“ na začátek a název fronty je odvozen rovněž z adresářové cesty, ale bez předpony „_noti“, je tím zajištěno, že MAIN může předat svůj název




Obrázek 23: Stavový diagram spuštění aplikace

fronty, vložením svého názvu do fronty HW driveru (jelikož zná její název) s příkazem pro jeho uložení. A taky může vytvořit reference pro notifikátory v MAINu. Po tomto procesu proběhne takzvaný Handshaking pro ověření, zda jsou oba komunikační kanály funkční.

Název fronty vypadá posléze takto: „D:\Universal test platform\HW Drivers\Keysight 34465A\Voltage from KS 34465A2.vi“ a notifikátoru: „noti_D:\Universal test platform\HW Drivers\Keysight 34465A\Voltage from KS 34465A2.vi“.

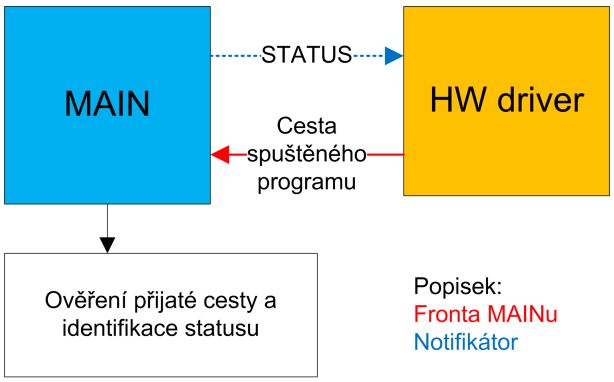



Obrázek 24: Komunikace

Spuštění měření probíhá následovně, skrz notifikátor se zašle příkaz „TRIGGER“, kterým HW driver reaguje provedením měření a uložením systémového času. Po naměření HW driver předá data do svého skriptu pro zpracování dat. Následně HW driver předá zpracovaný řetězec, který obsahuje čas změření a data do fronty MAINu spolu se svoji cestou, kde proběhne jejich následné uložení do měřícího logu. Samotná implementace řešení je rozebraná v další kapitole.

### 4.7.1 Handshaking

Česky by se dalo přeložit jako potřesení ruky. Je to automatizovaný proces známý především v telekomunikacích, při kterém dochází k vyjednávaní mezi dvěma účastníky a výměně dat. Nejčastěji dochází k zavádění komunikačních spojení, vyjednání protokolu a signalizací, že jsou obě zařízení připravena pro zahájení komunikace a výměnu dat. V navrhnuté platformě je využita jednodušší implementace tohoto komunikačního nástroje a slouží pouze k ověření průchodnosti komunikačních kanálů.




Obrázek 25: Handshaking

Komunikace mezi MAINem a HW driverem probíhá skrz notifikátor a unifikované příkazy. Jedním z nich je i příkaz „STATUS“ zaslaný ze strany MAINu do HW driverů. Po zaslaní příkazeu obslužný HW driver odpoví MAINu skrz jeho frontu příkazem „driverStatus“, a zašle v datech svoj název adresářové cestoy. MAIN má následně možnost ověřit, zda spuštění HW driveru proběhlo úspěšně, jelikož zná jeho adresářovou cestu a může si ji porovnat s obdrženou cestou. Pokud se cesty shodují, tak oba směry komunikace jsou funkční. Po ověření komunikace je HW driver shledán, že je schopný reagovat na příkazy a taky je schopen na ně odpovědět skrz sdílenou frontu MAINu.

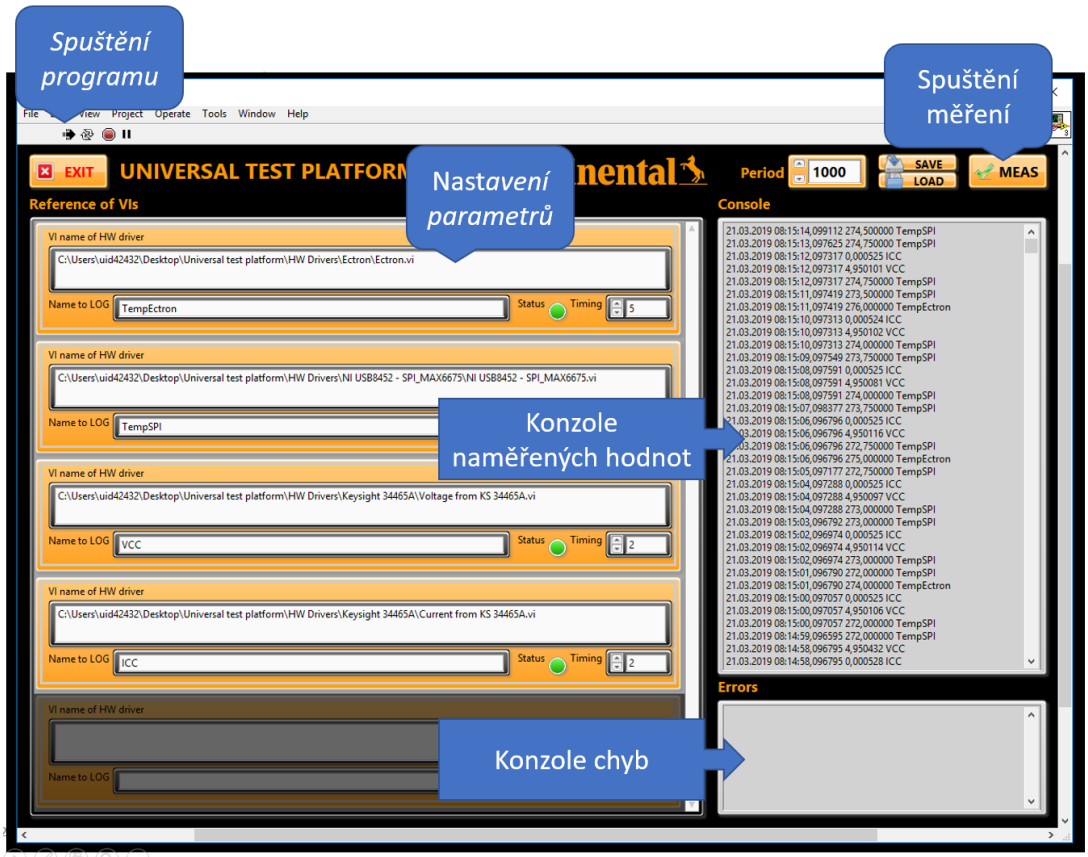
# 5 Implementace SW dle návrhu ve vývojovém prostředí LabVIEW

## 5.1 Uživatelské prostředí

Implementace následně probíhala ve vývojovém prostředí National Instruments LabVIEW 2016. Uživatelské prostředí a celá filozofie programu byla navržena s myšlenkou co nejednoduší obsluhy.

### 5.1.1 Čelní panel MAINu

V prostředí MAINu v položce „Reference of VIs“ obsluha testeru nastaví parametry jednotlivých HW driverů, jako je adresářová cesta pro spuštění, název přístroje pro měřící log, a časování. Časování je závislé na nastavené základní periodě vzorkování. U každého přístroje se nastavuje její násobek. Pokud je tedy nastaveno 1000ms a u přístroje v „Timing“ hodnota 5, je přístroj volán každých 5 sekund. Na přiloženém obrázku č. 26, jde vidět pět možných HW driverů. Jejich počet není omezen a v případě zaplnění je možné pomocí posuvníku doplňovat další. Konfiguraci daného testu je následně možné uložit pomocí tlačítka s názvem „SAVE“ nebo načíst pomocí tlačítka „LOAD“.




Obrázek 26: Grafické rozhraní MAINu

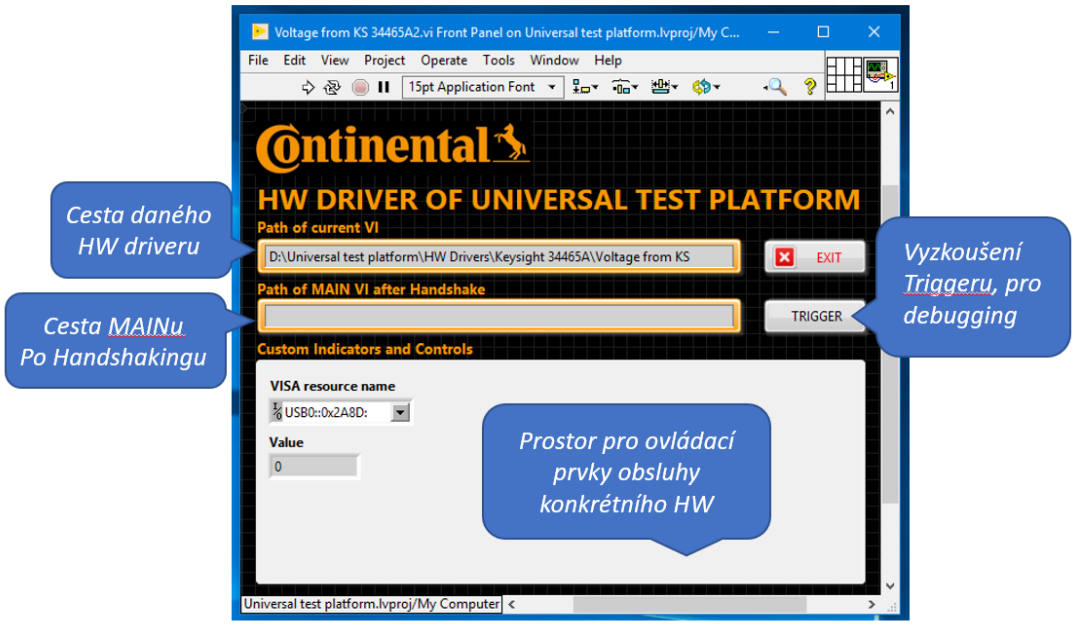
Program dále v levé části má konzoly pro výpis logu chyb a měření. Jejich obsah je zálohován do textových souborů pro možnou analýzu měření a případné zpracování chyb.

### 5.1.2 Nastavení Mainu

Pro spuštění testu a měření jsou potřeba tyto kroky:

1. Nastavení parametrů
    - Nastavení cesty HW driveru
    - Nastavení násobku hlavního časovače, pro interval volání
    - Nastavení Názvu pro identifikaci v logu
2. Spuštění programu
    - Spuštění HW driverů, dle nastavených cest
    - Ověření komunikace pomocí handshakingu
    - Indikace statusu spojení
3. Spuštění měření

### 5.1.3 Čelní panel HW driveru



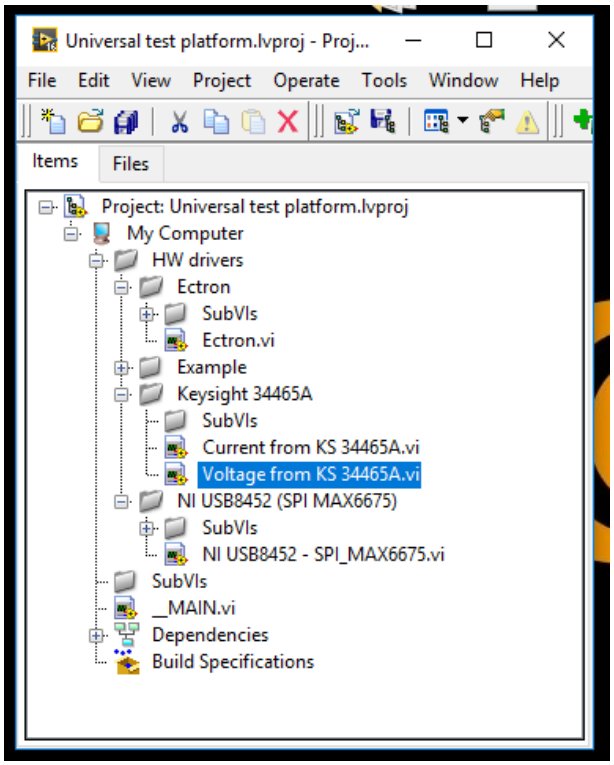

Obrázek 27: Grafické rozhraní HW driveru

V připravené šabloně jsou obsaženy všechny nezbytné položky. Jednou z nich je po spuštění ukázaní aktuální adresářové cesty, kterou je možné překopírovat do MAINu. Pokud je HW driver otevřen a je spuštěn vzdáleně z MAINu, jde vidět cesta MAINu a lze ověřit její správnost. Poslední položkou je tlačítko „TRIGGER“ pro simulaci vzdáleného volání MAINu, toto tlačítko slouží převážně k ladění HW driveru a ověření jeho funkčnosti.

Níže se nachází vyhrazený prostor pro umístění uživatelských prvků pro daný HW driver, převážně se jedná o nastavení reference pro VISA, DAQ nebo sériovou linku. Dále je možné umístit indikátor změřené hodnoty, případně jiný prvek, který bude potřeba.

## 5.2 Adresářova struktura projektu

Struktura projektu je členěna, jak je z obrázku č. 28 patrné, na jednotlivé HW drivery. Ty jsou členěny ve složkách. Tato struktura by měla být zachována pro budoucí orientaci a snadné přenášení HW driverů mezi počítači.




Obrázek 28: Struktura v projektu

## 5.3 MAIN

Hlavní program se skládá ze tří smyček. První zajišťuje obsluhu uživatelského prostředí, druhá samotné řízení programu pomocí stavového automatu a třetí časovací smyčku pro zasílání příkazů do HW driverů.

Obsluha uživatelského prostředí je založena na takzvané událostně řízené struktuře, která vykonává svůj kód při podnětu ze strany uživatele. Pokud uživatel provede podnět, jako je například stisk tlačítka, zavolá se odpovídající „case“ a vloží se příkaz do řídící fronty. Zaslaný příkaz je následně zpracován ve stavovém automatu hlavní smyčky. Příkazy v stavovém automatu jsou kromě uživatelského prostředí volány i z HW driveru. Odkud je volán stav pro uložení dat. Jeden stav může mít pod sebou více dílčích stavů, které následně volá. Na konci řídící smyčky je skript pro ukládání chyb, který danou chybu uloží do logu a následně ji smaže, tak aby bylo možné pokračovat v běhu programu. Poslední smyčkou je smyčka, ve které se volají jednotlivé HW drivery příkazem „TRIGGER“ v nastavených časech. Tato struktura je zobrazena detailněji na obrázku číslo 30.

## 5.4 HW driver

Je postaven na podobné architektuře jako MAIN. Rovněž obsahuje smyčku pro obsluhu uživatelského prostředí a hlavní řídicí strukturu. Ty jsou propojeny skrz frontu. Struktura fronty je identická jako u MAINu a je níže rozepsána. Poslední smyčkou je smyčka na měření, která je řízena notifikátorem a má pár přesně definovaných stavů, které zajišťují inicializaci měření, měření, ukončení měření a handshaking.

Předávání dat mezi měřící strukturou a hlavní smyčkou je skrz frontu. V hlavní smyčce v několika stavech dojde postupně k zpracování dat, kde dojde k zpracování měření a následnému zaslaní do MAINu je skrz frontu. Těchto stavů může být víc a je na budoucím programátorovi, jak daný HW driver napíše pro daný kus měřícího přístroje nebo DUT a požadavku na zpracování dat. Detailní ukázka kódu je na obrázku číslo 31.

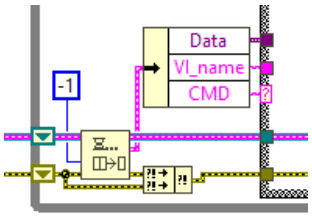
## 5.5 Fronta MAINu a HW driveru

Proměnná, která je předávaná pomocí fronty, je ve formě clusteru (struktury) a obsahuje tyto položky:

- CMD (string)
- VI_name (string)
- Data (variant)

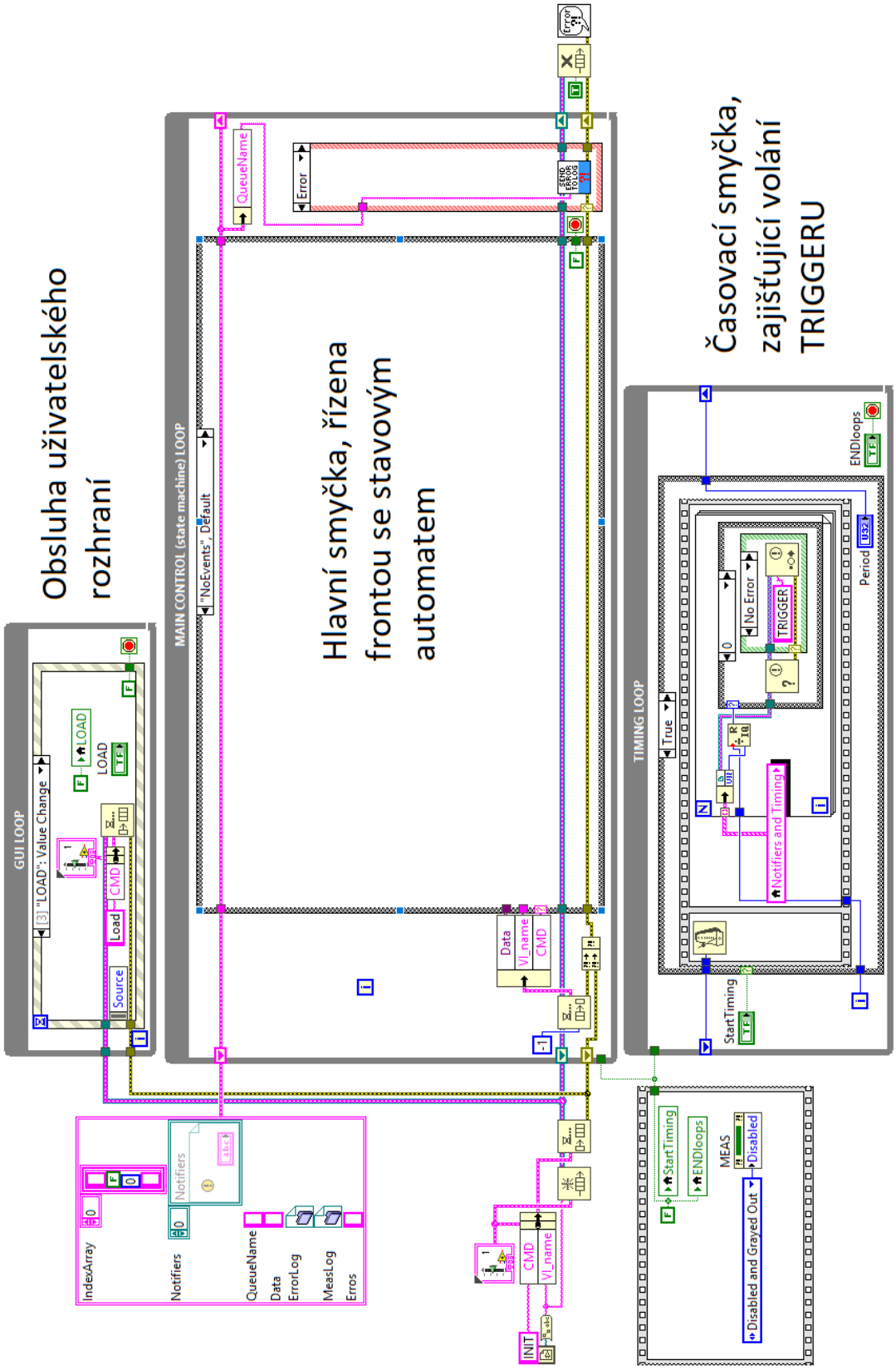
Příkazem CMD se volá odpovídající Case. Řetězec znaků byl zvolen z toho důvodu, že se v návrhu počítalo s voláním této fronty i mimo toto VI, tak aby bylo možné mít obecné příkazy a speciální příkazy. U ostatních přístupů, jako je například enumerátor není možné mít jednoduše

nedefinované speciální příkazy. V případě speciálního příkazu který daná struktura nezná, se zavolá výchozí Case „NoEvents“, ve kterém se nic nevykoná. Tento přístup je dost modulární a univerzální.

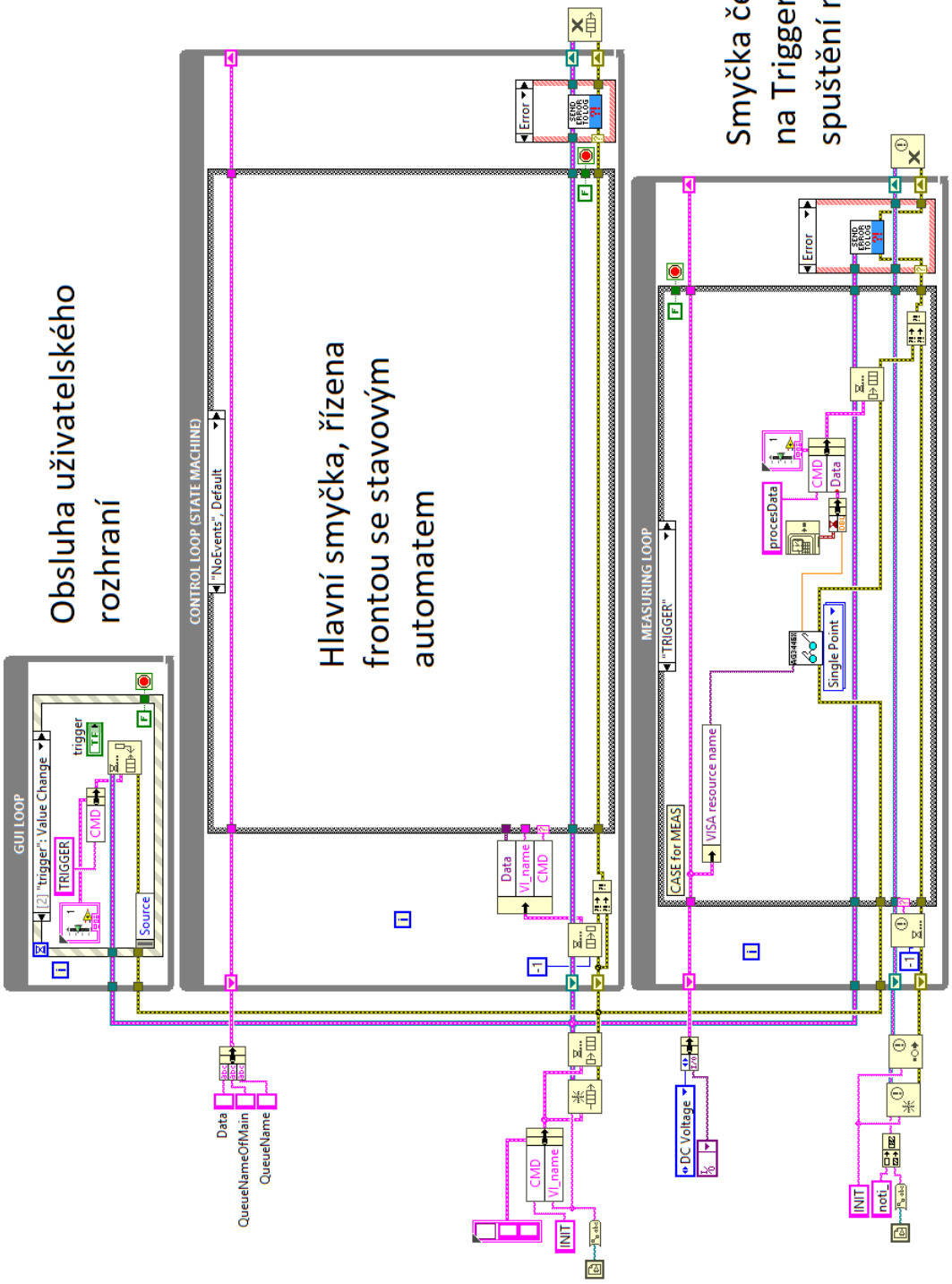



Obrázek 29: Vyzvednutí dat z fronty a ovládání stavového automatu

Posledními položkami je VI_name pro předávání názvu konkrétního zdroje odkud je VI spuštěno. Pokud se jedná o příkaz například z HW driveru, je možné identifikovat zdroj, odkud příkaz byl zaslán. Data jsou ve formátu variant a je možné tedy skrz něj přenášet libovolný typ dat.




Obrázek 30: Kód MAINu

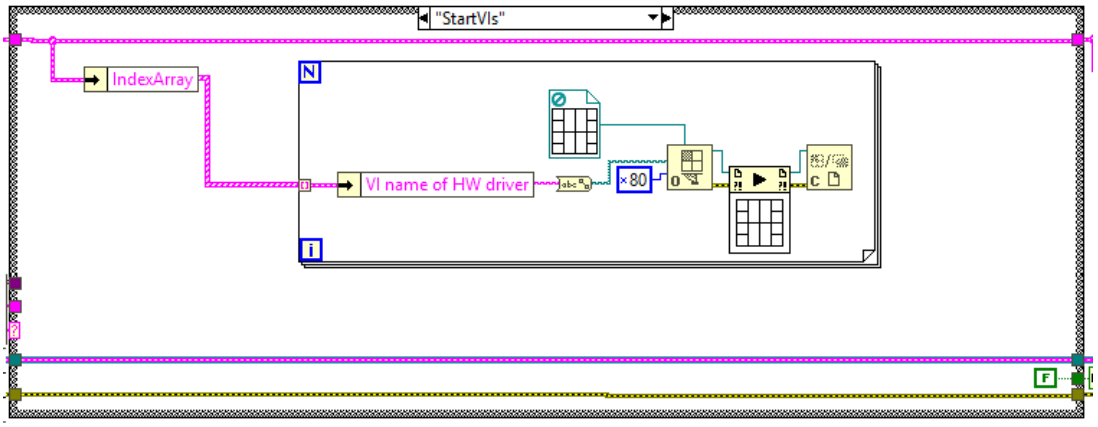



Obrázek 31: Kód HW driveru

## 5.6 Ukázky kódu

### 5.6.1 Dynamické volání VIs

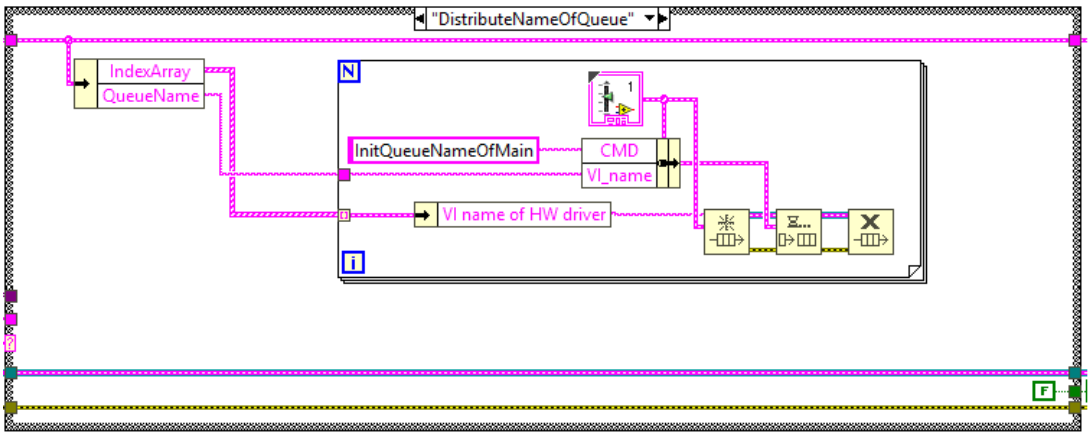
Při dynamickém spuštění HW driverů jsou v prvním kroku načteny jednotlivé cesty a následně ve smyčce spuštěny na pozadí. Spuštění obnáší nastavit masku VI, tedy jaké obsahuje vstupy a výstupy a nastavení spuštění. Nastavená hodnota 0x80 v „Open VI Reference“ definuje přípravu pro volání a následné zapomenutí po zavolání. Je vhodná pro asynchronní volání a není potřeba z VI v budoucnu požadovat výstupy, myšleno skrz rozhraní konektoru.




Obrázek 32: Dynamické spouštění VIs HW driverů

### 5.6.2 Předání názvu fronty

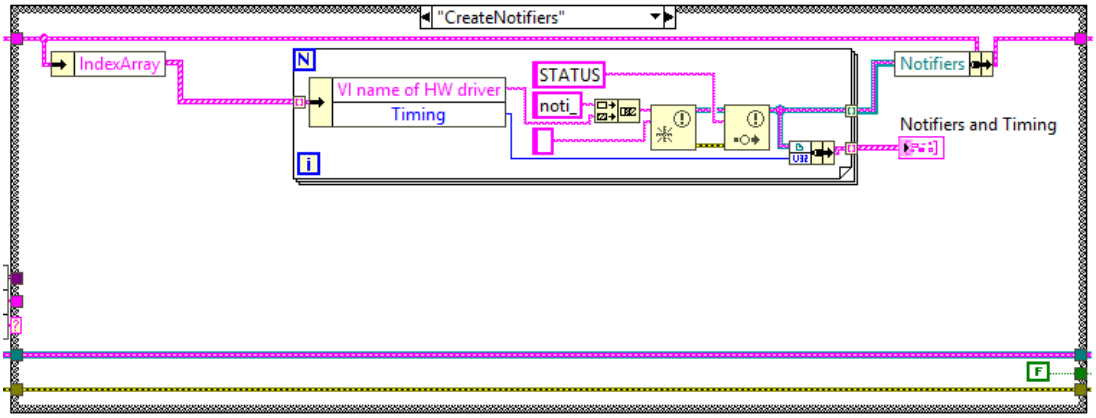
Tento kus kódu se zavolá po spuštění VI MAINu. Skrz fronty jednotlivých VI se následně zašle příkaz pro uložení fronty s názvem fronty do HW driverů. Následně jednotlivé HW drivery mají možnost odpovídat skrz tuto přednastavenou frontu MAINu. Toto je možné, jelikož název fronty je vždy tvořen adresářovou cestou.



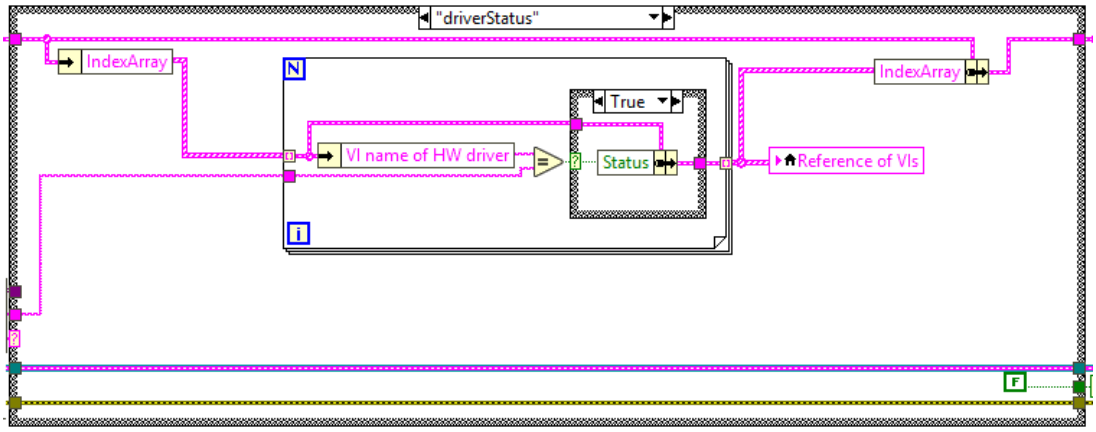

Obrázek 33: Zaslaní jména fronty MAINu do fronty HW driveru

### 5.6.3 Vytvoření notifíkátorů a Handshaking

Název notifikátoru pro zasílání příkazů z MAINu a HW driveru je rovněž odvozen od cesty HW driveru s přidáním _noti na začátek. Následně je možné provést ověření komunikace zasláním příkazu „STATUS“ skrz notifikátor. Tento stav se vykonává po zaslaní jména fronty MAINu, a je tedy možné, aby HW driver odpověděl zavoláním stavu „driverStatus“ a zaslaním skrz frontu svoji adresářovou cestu. Následně v tomto stavu je možné provést porovnání a indikování, jestli je komunikace oběma směry průchozí.



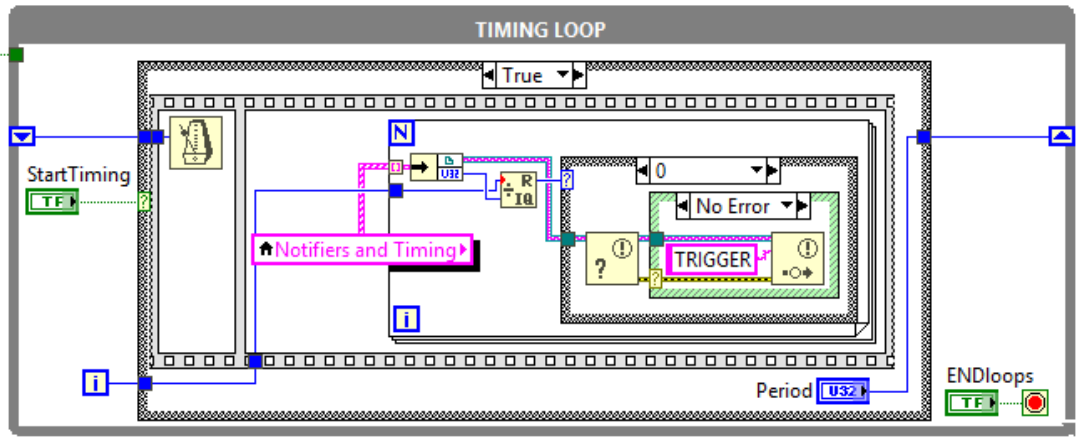

Obrázek 34: Vytvoření notifikátorů a zaslání příkazu „STATUS“ pro Handshaking




Obrázek 35: Ověření vrácené cesty z HW driveru

### 5.6.4 Časování a Trigger

Tato část kódu MAINu je periodicky volána dle nastaveného časovače funkce „Wait Until Next ms Multiple“, pokud zrovna není zapnuté měření a nevykonává se žádný kód. V opačném případě jsou jednotlivé HW drivery periodicky volány. To je zajištěno modulem, kdy je driver zavolán pokud je splněna podmínka, že vnitřní čítač smyčky má nulový zbytek po celočíselným dělení nastaveného časovače. Následně je zaslán příkaz „TRIGGER“ z MAINu do HW driveru pro zahájení měření.

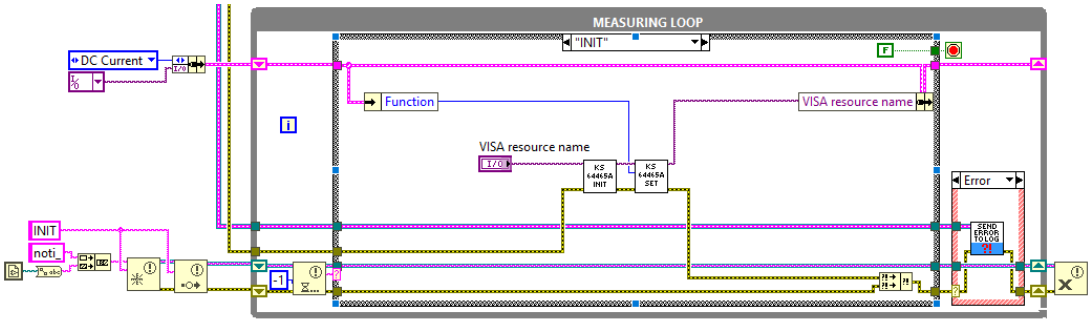



Obrázek 36: Timer a trigrování

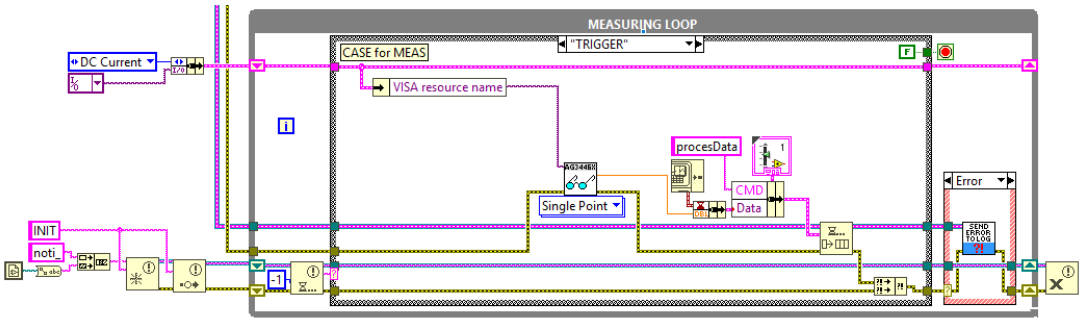
### 5.6.5 Stavy ve stavovém automatu v měřící smyčcce HW driveru

V měřící smyčce je omezený počet stavů řízených notifikátorem. Ve výčtu se jedná o stav inicializace, který se volá interně v HW driveru po jeho zapnutí. Má za úkol vytvořit komunikační spojení s přístrojem a provést jeho nastavení například skrz rozhraní VISA, DAQ atd.

Další stav pro Trigger je volán buď interně pro účely ladění nebo v průběhu měření z MAINu z časovací smyčky. Při měření dochází pouze k získaní naměřené hodnoty a systémového času, tyto data se dále předávají do stavového automatu pro zpracování skrz vnitřní frontu HW driveru.



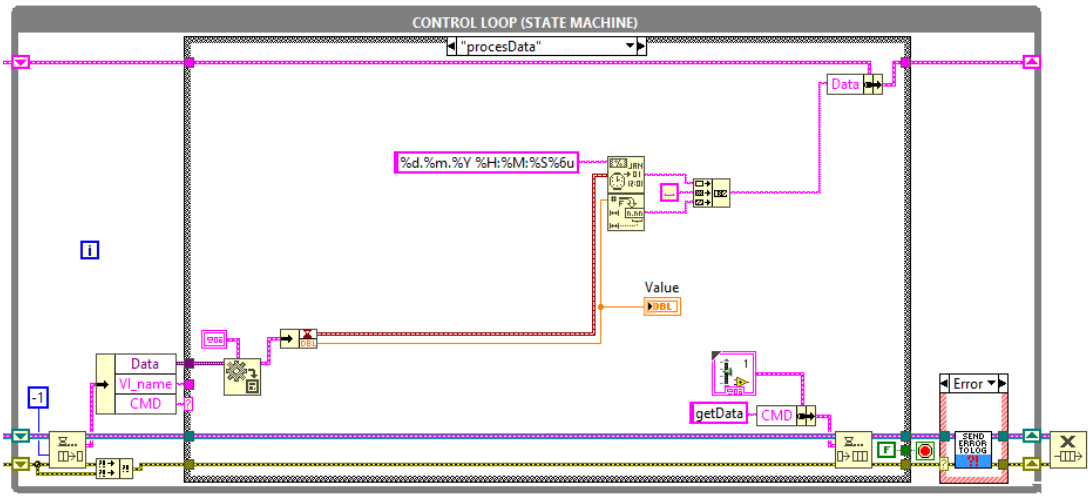

Obrázek 37: HW driver, inicializace přístroje



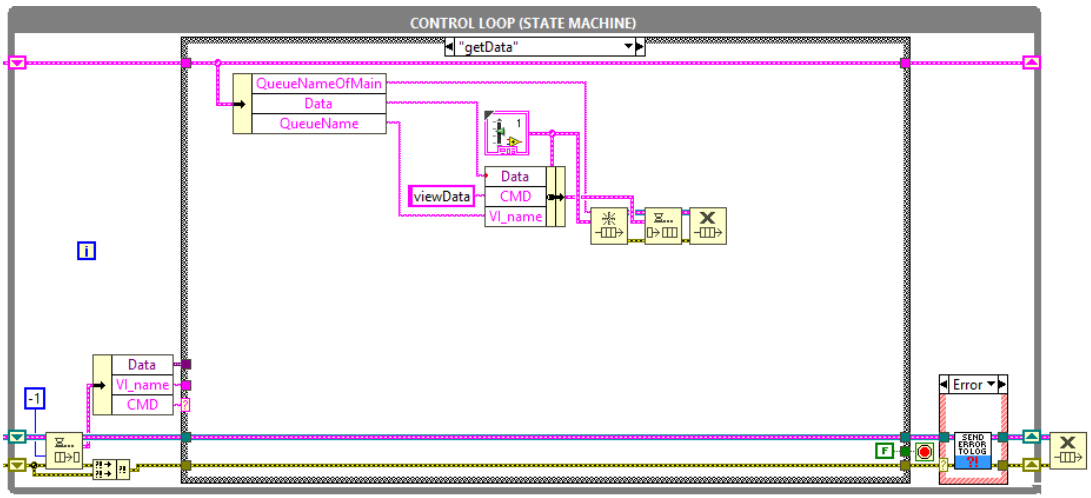

Obrázek 38: HW driver, trigger

### 5.6.6 Stavy zpracování a zaslaní dat uvnitř HW driveru

Zaslaná data z měřící smyčky jsou v této struktuře v prvním kroku zpracována, ve stavu pro zpracování dat je prostor na umístění libovolného zpracování dat. Musí se pouze zachovat formát zasílání času a výstup v podobě textového řetězce, jelikož výsledný řetězec je v následujícím stavu zasílán do fronty MAINu, kde v této podobě dojde k jeho uložení. Spolu s Daty se zasílá adresářová cesta HW driveru tak, aby v MAINu bylo možné přijatá data identifikovat a přiřadit nastavený název pro uložení do měřícího logu.



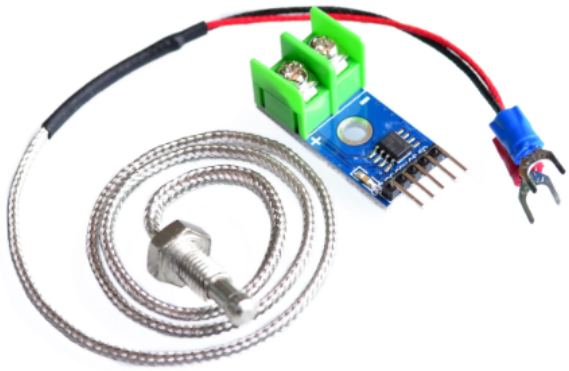

Obrázek 39: HW driver, Zpracování dat




Obrázek 40: HW driver, Zaslaní dat do MAINu

# 6 Verifikace a testování SW architektury na vybraném senzoru

Test byl proveden na senzoru MAX6675. Jedná se o 12bit digitální převodník K – termočlánku s kompenzací studeného spoje, který je schopen měřit teplotní rozsah od 0°C do +1024°C. Pro získaní výsledné teploty je nutné přijatou 12bit hodnotu v rozsahu 0 až 4096 vydělit čtyřmi, výsledná přesnost vrácené hodnoty je ±0.25°C. [15]



Obrázek 41: MAX6675 s termočlánkem

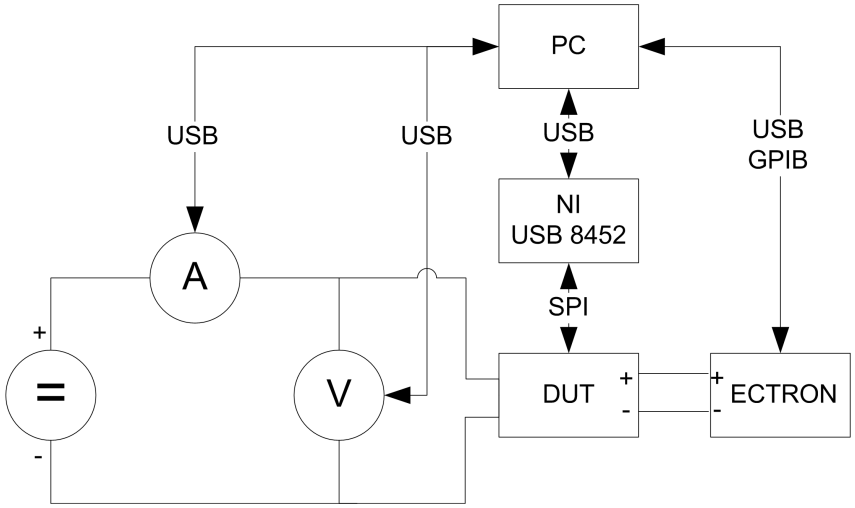
Přístroje byly zapojeny do testovací sestavy, kterou tvořily tyto přístroje: testovaný senzor MAX6675, NI USB-8552, Keysight 34465A jako Voltmetr, Keysight 34465A jako Ampérmetr, Simulátor termoelektrického napětí ECTRON 1140, převodník GPIB na USB a PC s univerzální testovací architekturou.

Pro každý přístroj byl napsaný samostatný driver, v MAINu potom byly nastaveny tyto parametry:

Tabulka 1: V MAINu byly pro HW driver nastaveny tyto parametry

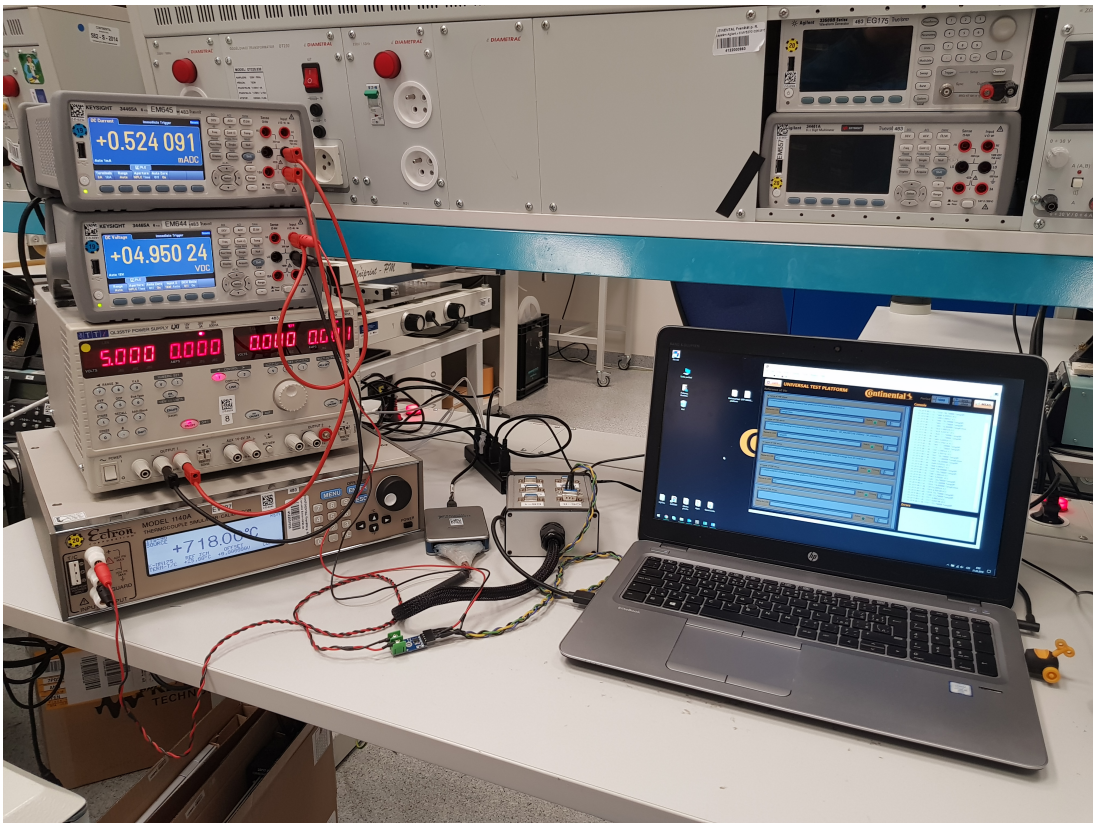
| **Zařízení** | **Použití** | **Název v logu** | **Perioda volání** |
|---|---|---|---|
| NI USB-8452 | Převodník SPI na USB | TempSPI | 1s |
| Keysight 34465A | Ampermetr | ICC | 2s |
| Keysight 34465A | Voltmeter | VCC | 2s |
| ECTRON 1140 | Simulator termoelektrického napětí | TempEctron | 5s |

Přístroje byly volány s různou periodou tak, aby bylo možné ověřit schopnost synchronizace a při zpracování mít možnost porovnat hodnoty mezi sebou. Měření probíhalo následovně, během tohoto demonstračního měření bylo za úkol ověřit přesnost MAX6675, ze kterého se každou sekundu sbíral vzorek naměřené hodnoty. Každé dvě sekundy probíhalo měření napájecího napětí a proudu a každých pět sekund probíhala změna na simulátoru termoelektrické napětí ECTRON,




Obrázek 42: Grafické znázornění zapojení testovací sestavy

přístroj byl kalibrovaný a byl vzat jako reference která postupně o 1°C zvyšovala teplotu v rozsahu 0°C - 1024°C. Tímto byl proměřen celý rozsah senzoru a bylo možné porovnat hodnoty s kalibrovaným simulátorem ECTRON, který byl vzat jako reference.

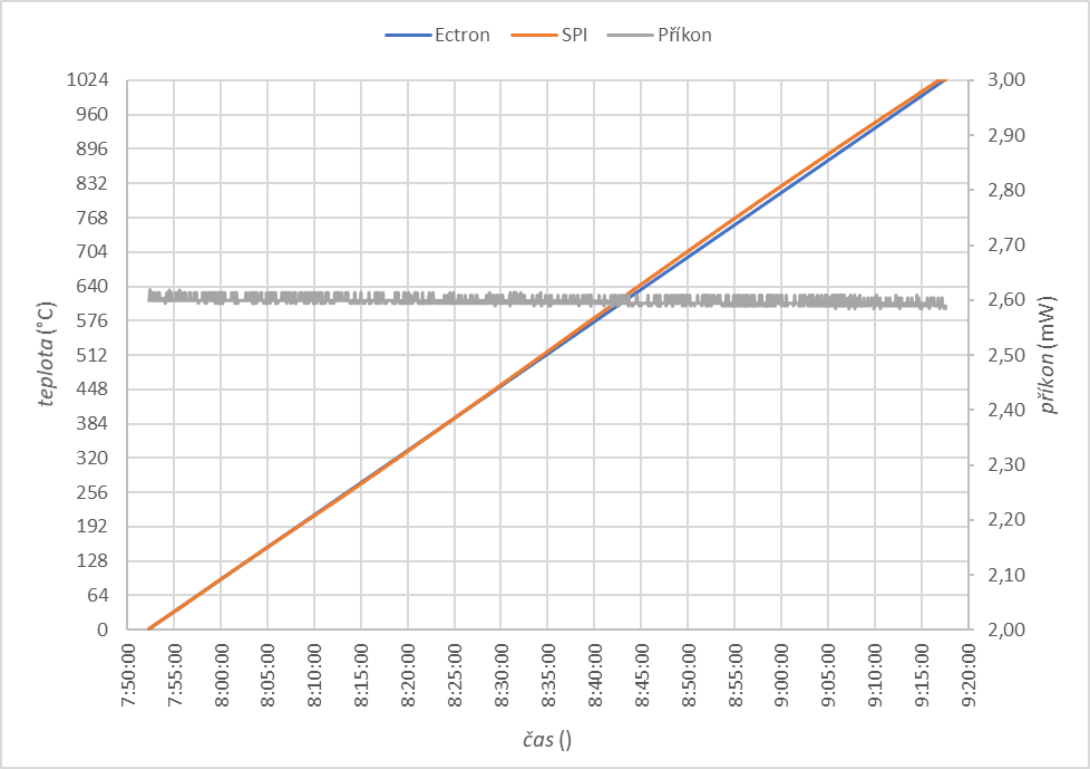



Obrázek 43: Testovací sestava

## 6.1 Zpracování naměřených hodnot

```
DATE TIME VALUE NoD
21.03.2019 07:52:16,437090 1,000000 TempEctron
21.03.2019 07:52:16,437090 1,250000 TempSPI
21.03.2019 07:52:16,803304 2,500000 TempSPI
21.03.2019 07:52:16,437090 4,950154 VCC
21.03.2019 07:52:16,437090 0,000525 ICC
21.03.2019 07:52:17,802959 2,250000 TempSPI
21.03.2019 07:52:17,802959 4,950101 VCC
21.03.2019 07:52:17,802959 0,000525 ICC
21.03.2019 07:52:18,801958 2,250000 TempSPI
21.03.2019 07:52:19,801300 2,250000 TempSPI
21.03.2019 07:52:19,801300 4,950060 VCC
21.03.2019 07:52:19,801300 0,000526 ICC
21.03.2019 07:52:20,799932 2,000000 TempEctron
21.03.2019 07:52:20,799932 2,000000 TempSPI
21.03.2019 07:52:21,798709 3,500000 TempSPI
21.03.2019 07:52:21,798709 0,000528 ICC
21.03.2019 07:52:21,798709 4,950328 VCC
21.03.2019 07:52:22,797593 3,500000 TempSPI
21.03.2019 07:52:23,796407 3,250000 TempSPI
21.03.2019 07:52:23,796407 0,000529 ICC
21.03.2019 07:52:23,796407 4,950292 VCC
21.03.2019 07:52:24,795836 3,500000 TempSPI
```

Data byla následně zpracována v tabulkovém procesoru Microsoft Excel 365 ve verzi 1808. Pro zpracování dat bylo v prvním kroku nutné separovat jednotlivé položky dle identifikátoru, v logu vedeném jako „NoD“ z anglických slov Name of Device, neboli jméno zařízení. Po tomto kroku byla jednotlivá data spolu s časy ve sloupcích a bylo možné vykreslit v XY grafu závislost naměřených hodnot v čase s různou dobou vzorkování. Pokud byla doba stejná je možné u dat rovnou provést zpracování, například vypočítat příkon vynásobením napájecího napětí a proudu. Toto je možné díky synchronizaci v měření, kdy přístroj pro měření proudu a přístroj pro měření napětí vrátí naměřené hodnoty se stejnou časovou značkou a je následně možné provést porovnání a výpočet.




Obrázek 44: Průběh měřené a generované teploty s vypočteným příkonem

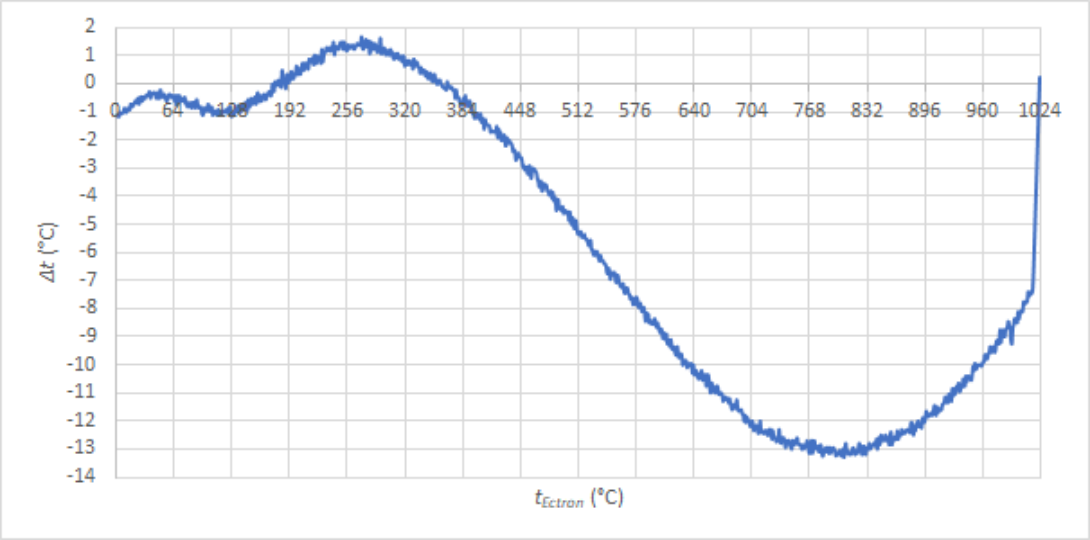
Pro zpracování dat s jinou dobou vzorkování bylo nutné v Excelu provést následující úkony. V prvním kroku pomocí funkcí KDYŽ (IF), SVYHLEDAT (VLOOKUP) a JE.NEDEF (ISNA) spárovat hodnoty s větší vzorkovací periodou s hodnotami s menší periodou, tak aby na stejném řádku byla hodnota změřená ve stejný čas.

Tabulka 2: Porovnání dat s různou vzorkovací periodou

| TIME | TempSPI | TempEctron |
|---|---|---|
| 7:52:16 | 1,25 | 1 |
| 7:52:17 | 2,5 | |
| 7:52:18 | 2,25 | |
| 7:52:19 | 2,25 | |
| 7:52:20 | 2,25 | |
| 7:52:21 | 2 | 2 |
| 7:52:22 | 3,5 | |
| 7:52:23 | 3,5 | |
| 7:52:24 | 3,25 | |
| 7:52:25 | 3,5 | |

Následně bylo možné provést průměr pěti hodnot a odečíst je od referenční hodnoty Ectronu, tyto hodnoty následně od sebe odečíst a vytvořit korekční křivku závislosti absolutní chyby

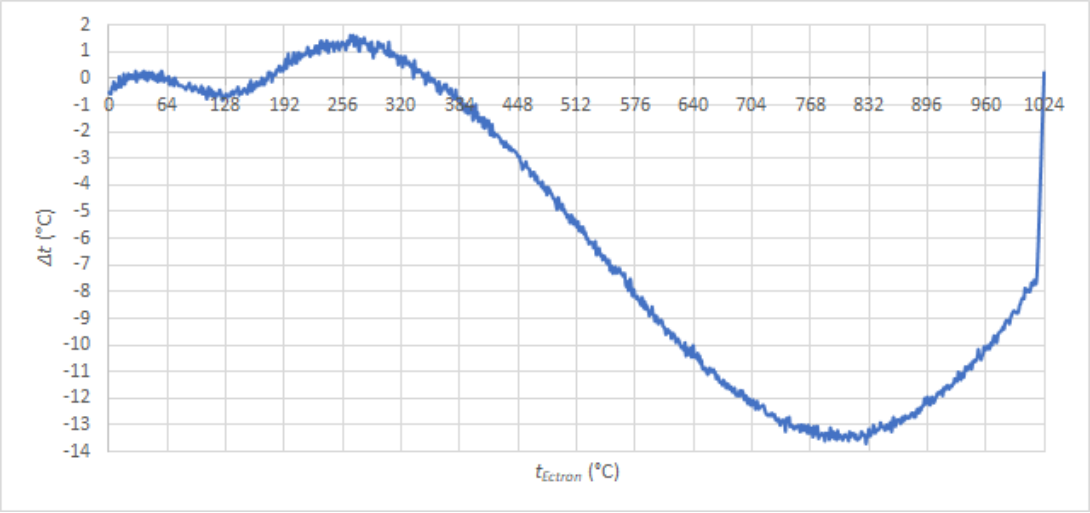
měřené teploty pomocí MAX6675 a porovnat vůči kalibrované referenci Ectronu 1140 a vykreslit v grafu.




Obrázek 45: Korekční křivka prvního senzoru

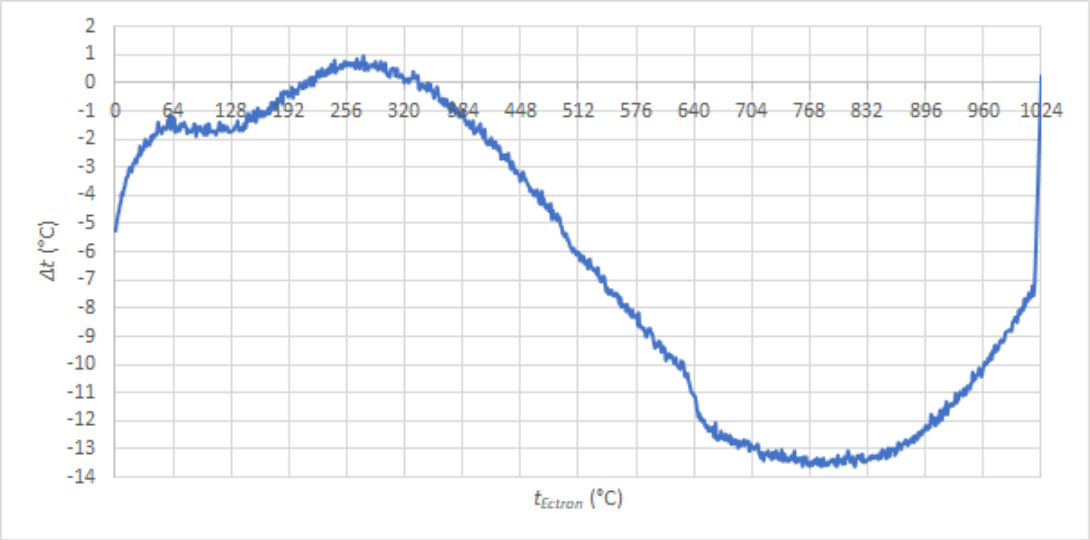
Měření bylo provedeno pro celkově tři senzory. Korekční křivka byla kontrolována s technickou specifikací MAX6675, který má uváděnou abloslutní chybu ±9°C pro rozsah teplot od 0°C do +700°C a ±19°C pro rozah teplot od 700°C do +1000°C při 5V napájecím napětí. [15]

První senzor je mimo tento stanovený rozsah od 611°C do 700°C, druhý senzor je mimo rozsah od 598°C do 700°C, a třetí senzor je mimo tento rozsah od 595°C do 700°C, kde je absolutní chyba vyšší než stanovených 9°C.




Obrázek 46: Korekční křivka druhého senzoru

Cílem této verifikace testovacího SW architektury nebylo ověření samotného senzoru MAX6675, ale schopnost SW architektury sbírat data z jednotlivých přístrojů s různým vzorkováním, pro-




Obrázek 47: Korekční křivka třetího senzoru

vádět synchronizaci a v poslední řadě ukázat, že je následně možné zpracovat uložená data do podoby, s kterou je možné jednoduše pracovat pomocí obyčejného tabulkového procesoru. A vyhodnotit výsledky měření.

# 7 Zhodnocení dosažených výsledků a závěr

V diplomové práci jsem se zabýval vývojem softwarové univerzální architektury pro oblast testování senzorů pro automobilový průmysl. Na začátku práce byla rozebrána problematika programování v LabVIEW a možné přístupy z pohledu programovacích vzorů. Dále jsem se zabýval problematikou testování senzorů pro automobilový průmysl a uvedl jaké důvody vedly společnost Continental k vývoji daného testeru.

V průběhu návrhu bylo realizováno několik konceptů, ze kterých vyšel výsledný návrh. Ten je tvořen ze dvou základních částí, takzvaného MAINu a HW driveru. MAIN má za úkol obsluhovat a sbírat data z jednotlivých HW driverů, ty si můžeme představit jako nezávislé zapouzdřené programy zajišťující sběr, zpracování a zasílání dat. HW driver může fungovat samostatně, ale ve spojení s MAINem dokáže reagovat na obecné příkazy a vracet unifikovaný výstup. Počet HW driverů není z pohledu navrhnuté architektury omezen. Jednotlivé HW drivery je možné volat s různou periodou, která je násobkem hlavní taktovací periody. Výsledný univerzální tester kromě modularity dovoluje volat několik HW driverů synchronně, je tedy možné porovnávat závislost naměřených hodnot. SW struktura, je vytvářena jako platforma pro budoucí modifikace, která zajišťuje základní úkony jako je komunikace mezi moduly, synchronní volání HW driverů a ukládání dat. Součástí práce bylo napsaní i několika demonstračních HW driverů. Pro budoucí rozšíření bude programátorovi stačit použít poskytnutou šablonu a doprogramovat sběr a zpracování dat. Šablona již poskytuje všechny nezbytné prostředky, jako je komunikace s MAINem, nezávislost měření od zpracování dat a zaznamenávání dat a chybových hlášek.

Univerzální SW struktura byla ověřena na třech teplotních senzorech MAX6675, které měří teplotu pomocí termočlánku. Jednotlivé senzory byly otestovány v zapojení simulující reálné testování senzorů vyvíjených ve společnosti Continental. Během měření byla sbíraná komunikace se senzorem, měřeno napájecí napětí a proud a měněné podmínky měřené teploty. Pro účely měření byl použit místo reálného termočlánku kalibrovaný simulátor termoelektrické napětí, senzor byl proměřen v jeho pracovním rozsahu od 0°C do 1024°C. Pro demonstrační účely, byly přístroje volány s různou vzorkovací periodou. Výsledný log z měření byl následně zpracován v tabulkovém procesoru MS Excel, kde byla vykreslena závislost měřených hodnot na čase a vypočtena korekční křivka.

Diplomová práce splňuje všechny body zadání, a rozšiřuje je o možnost synchronizace. Výsledný tester bude využíván ve společnosti Continental Powertrain Czech Republic s.r.o., pro inženýrské a validační testy.

# Literatura


[1] RF Design Services from Cambridge RF [online]. [cit. 2019-03-16]. Dostupné z: https://www.cambridgerf.com/design_services.htm

[2] lying Probe Tester [online]. [cit. 2019-03-16]. Dostupné z: https://acculogic.com/products/flying-probe-testers/flying-scorpion-fls980

[3] Aplikování sběrnice CAN [online]. [cit. 2019-03-16]. Dostupné z: https://vyvoj.hw.cz/navrh-obvodu/rozhrani/aplikovani-sbernice-can.html

[4] Achieving Fast SENT Message Response with A1346 Linear Hall-Effect Sensor ICs on Shared Bus [online]. [cit. 2019-03-16]. Dostupné z: https://www.allegromicro.com/en/Design-Center/Technical-Documents/Hall-Effect-Sensor-IC-Publications/AN296131-Fast-SENT-Message-Response-A1346-Shared-Bus.aspx

[5] Pulse-width modulation. Wikipedia [online]. [cit. 2019-04-28]. Dostupné z: https://en.wikipedia.org/wiki/Pulse-width_modulation

[6] Effective labview programming. 1. Allendale, NJ: NTS Press, 2013. ISBN 978-1934891087.

[7] BITTER, Rick, Taqi MOHIUDDIN a Matt NAWROCKI. LabView advanced programming techniques. 2nd ed. Boca Raton, FL, c2007. ISBN 978-084-9333-255.

[8] National Instrument: What Is LabVIEW? [online]. [cit. 2019-01-13]. Dostupné z: http://www.ni.com/cs-cz/shop/labview.html

[9] Elektronické studijní materiály: Návrhové vzory. Mendelova univerzita v Brně [online]. [cit. 2019-01-12]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=32198

[10] National Instrument: Application Design Patterns: State Machines [online]. [cit. 2019-01-13]. Dostupné z: http://www.ni.com/white-paper/3024/en/

[11] National Instrument: Design Patterns in NI LabVIEW: Developer Days 2009 [online]. [cit. 2019-01-13]. Dostupné z: ftp://ftp.ni.com/pub/events/labview_dev_ed/2009/labview.pdf

[12] National Instrument: Application Design Patterns: Master/Slave [online]. [cit. 2019-01-13]. Dostupné z: http://www.ni.com/white-paper/3022/en/

[13] National Instrument: Application Design Patterns: Producer/Consumer [online]. [cit. 2019-01-13]. Dostupné z: http://www.ni.com/white-paper/3023/en/

[14] National Instrument: Fundamental LabVIEW Design Patterns: Developer Days 2011 [online]. [cit. 2019-01-13]. Dostupné z: ftp://ftp.ni.com/pub/branches/germany/nidays_austria/03_anwendungsentwicklung/05_egger_ni.pdf

[15] Maxim Integrated [online katalogový list]. MAX6675 ©2014 [cit. 10.4.2019]. Dostupné z: https://datasheets.maximintegrated.com/en/ds/MAX6675.pdf